Novel Evolvable Hardware based on Memristors

By

Xinming Shi

A thesis submitted to
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
December 2023

ABSTRACT

Evolvable Hardware (EHW) often refers to an automated hardware design that draws inspiration from natural evolution. With the gradual slowdown of Moore's Law, EHWs based on traditional electronic devices are increasingly encountering physical limitations. Additionally, the separation of storage and computation in conventional hardware architectures poses further challenges. Memristors are a promising type of emerging electronic device that can alleviate these physical limitations and help to build better EHW. However, there has been limited research on how to evolve memristive hardware. Several challenges such as issues with existing circuit representations and algorithms used in memristor-based EHW are still open. This thesis studies novel EHW based on memristors, with the focus of improving circuit representations, improving evolutionary design processes, and exploring the potential applications of evolved memristive circuits. The main contributions of the thesis include:

- Two memristor-based reconfigurable architectures constructed by manipulating memristor properties.
- A novel circuit evolution approach using genetic programming, including a new circuit representation and a more efficient evolutionary algorithm.
- Two indirect circuit representations and corresponding evolutionary operators for reconfigurable memristive architectures.
- Novel evolutionary hardware techniques for reservoir computing applications, which can improve the accuracy and energy efficiency of the proposed EHWs.

ACKNOWLEDGMENTS

As a student of the Joint PhD Program offered by the University of Birmingham (UoB) and Southern University of Science and Technology (SUSTech), I would like to thank my supervisors, Dr. Leandro L. Minku in UoB and Prof. Xin Yao in SUSTech for their supervisions, and for the support from UoB and SUSTech during my studies at both universities. I am deeply grateful to to Dr. Leandro L. Minku, who provided me with valuable guidance on overcoming difficulties throughout my PhD. His attitude in researching has a great influence to me. I would also like to express my greatest gratitude to Prof. Xin Yao for his inspiration, patience and encouragement throughout my PhD. His enthusiasm, pure and unwavering pursuit to the research, meticulous adherence to the details have great influence on my own pursuits. This thesis would never be completed without their valuable suggestions and help.

I would like to thanks the thesis group committees in UoB, Dr. Rami Bahsoon and Dr. Shan He for their insightful comments and advice. I also thanks to brain-inspired computing meeting group members in SUSTech, Dr. Bo Yuan, Dr. Zilu Wang and other members for the in-depth discussions and valuable feedback.

I am very lucky to have the company of many good colleagues, researchers, and friends. Thanks to Jinyu Yang, Ruo Wang, Cheng Li, Nan Zhang, Linfang Zheng, Hao Tong, Xi Jia, Yanfang Wang and many other colleagues and friends in UoB. During my study in SUSTech, many thanks to Dr. Liyan Song, Dr. Changwu Huang and Dr. Weijie Zheng for their valuable help and suggestions. And I am also grateful to Jiashi Gao, Yinghua Yao, Shuyi Zhang, Yunce Zhao, Gan Ruan, Qingya Li, Zhi Cao, Shuxian Li, Han Zhang, and Dashan Gao and many other colleagues and friends in SUSTech. Thank all of you for the

help and support that have provided me in my research and life. With your companionship, my journey of pursing PhD has been filled with a lot of joy and warm.

Last but not the least, I would like to thank my family: my parents Qing Shi and Jing Gao, for all their love and encouragement. They supported me in all my pursuits and enabled me to study with no worries or pressure. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

Acronyms

ADC Analog to Digital Converter.

AGE Analog Genetic Encoding.

ANN Artifical Neural Network.

ARFIMA Autoregressive Fractionally Integrated Moving Average.

ASIC Application-Specific Integrated Circuit.

BF Best Fitness.

CMOS Complementary Metal Oxide Semiconductor.

CRJ Cycle with Regular Jump.

DAC Digital to Analog Converter.

DDE Delay Differential Equation.

DGR Dynamic Gesture Recognition.

DIM Dynamic Input Masking.

DJI Dow Jones Industria.

EA Evolutionary Algorithm.

EGG Evolutionary Graph Generation.

EHW Evolvable Hardware.

ES Evolution Strategy.

ESN Echo State Network.

FPGA Field-Programmable Gate Array.

GA Genetic Algorithm.

GE Grammatical Evolution.

GP Genetic Programming.

GRN Genetic Regulatory Network.

IS Input Scaling.

LM Levenberg Marquardt.

LR Leaky Rate.

LSM Liquid State Machine.

MBF Mean Best Fitness.

MDE Memristor-based Delay Element.

MSE Mean Square Error.

MWE Memristor-based Window Element.

NARMA Nonlinear Autoregressive Moving Average.

NVM Nonvolatile Memristor.

OPE Operational Amplifier.

PSO Particle Swarm Optimization.

RC Reservoir computing.

RLC Resistor-Inductor-Capacitor.

RMSE Root Mean Squared Error.

RMU Reconfigurable Memristive Unit.

RNN Recurrent Neural Network.

SRAM Static Random Access Memory.

TDR Time Delay Reservoir.

VRC Virtual Reconfigurable Circuit.

Chapter One

Introduction

Evolvable Hardware (EHW), is a scheme for automatically designing hardware systems through algorithms inspired by natural evolution (evolutionary algorithms) [69]. It has been a popular area of research since the 1990s due to its ability to adaptively change and provide desired functionality through reconfiguration, making it more flexible than ASIC-based systems [343, 295, 106, 303]. Figure. 1.1 outlines the research field of EHW, where EHW is at the confluence of Bio-inspired Hardware, Bio-inspired Software, and Conventional Computing. One could consider that there are two significant components of an EHW system, which are hardware and evolutionary algorithm. Therefore, limitations also exist from these two perspectives.

From the hardware perspective, Field-Programmable Gate Arrays (FPGAs) are commonly used reconfigurable hardware in EHW, but they suffer from high power consumption, limited computing density, and logical redundancy [256]. Memristors, on the other hand, offer the potential for implementing novel circuit architectures that can perform both operation and storage on a single device, as well as having small size and energy efficiency [333, 350]. Memristors were theoretically postulated by Chua in 1971 [48] and later Williams's team presented a resistance variable device as a memristor at HP Labs in 2008 [281]. Mem-

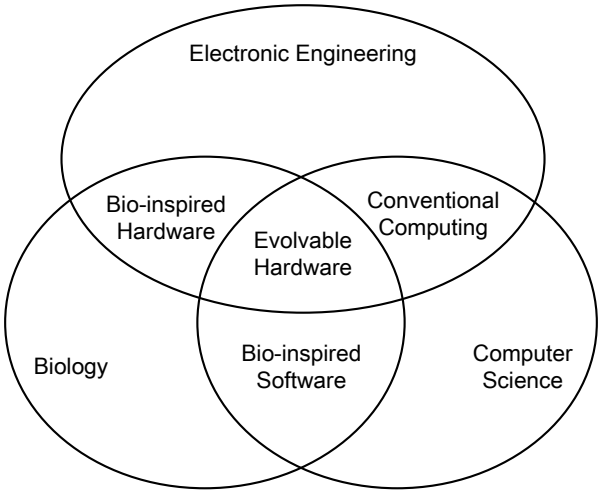


Figure 1.1: Research filed of EHW, which is redrawn from [302].

ristors are equipped with following characteristics: (1) Compared to traditional devices, memristors have nanoscale dimensions, making them suitable for use as synapses in neural networks, constructing large-scale and ultra-large-scale integrated neural networks; (2) Unlike electronic components with binary outputs, memristors are analog devices capable of continuous output, suitable for representing continuous variable weights; (3) As a passive device with non-volatility, memristors have low energy consumption characteristics; (4) Since the resistance value of memristors changes with magnetic flux and charge, they have the necessary memory and learning capabilities required as synapses in neural networks; (5) By constructing memristor crossbar arrays, the storage and information processing capabilities of neural networks can be greatly improved. Therefore, memristors are commonly used as synapses in neural networks, and their weight can be trained to further achieve the functions of the neural network. However, memristors also face some challenges and limitations that hinder their practical implementation and application, such as variation of memristor state and reliability issues that arise from the immature fabrication process and physical device limitations. Memristors also suffer from noise, large manufacturing variability, and short life cycles that hinder their practical implementation and application. These issues arise from the immature fabrication process and physical device limitations of memristors.

Despite the challenges and limitations of evolutionary algorithms and the technology-focused future of EHW that require acceleration and parallelization, we believe that memristors have the potential to create a novel and efficient EHW system. We aim to investigate the feasibility of using memristors to design a novel EHW system, harnessing the variations of memristors to implement different functions of the novel memristor-based EHW rather than considering them only as a negative features of memristors. In addition, brain-inspired computing, driven by the architecture and efficiency of the biological brain, presents a paradigm shift for enhancing computational processes [122]. Memristors, with their distinctive attributes of dynamics, analog computation, and nonvolatility, emerge as foundational components for these systems, replacing CMOS limitations with energy-saving and high-density advantages. Their adaptability as neural and synaptic analogs positions them as prime candidates for constructing advanced EHW. This pursuit is validated by the benchmarks selected for our research: memristive reservoir computing, pulse generation circuits, and computation circuits—all reflecting the core operations of brain-inspired processing and underscoring the practicality of memristor-based systems.

The algorithmic aspects of EHW also present challenges, including issues with existing circuit representations and algorithms used in EHW. The primary concern in EHW is the representation of the circuit, and there are several problems associated with the current circuit representation methods. These include inefficient transformation to the circuit netlist[1] and a lack of design ability, as explained by Koza et al. [150], Chang et al. [38], and Lohn et al. [178]. In terms of evolutionary algorithms, Genetic Programming (GP) [148] is a popular approach for circuit evolution. However, GP also faces challenges, such as an inefficient evolution process based on random evolutionary operators and the issue of bloat. The bloat issue of GP refers to the gradual increase in the size of the programs

[1]A circuit netlist is a text file that describes the circuit in terms of components, nodes and connections, which is used as input for the Simulation Program with Integrated Circuit Emphasis (SPICE) [231] software to perform the simulation of electrical circuits, enabling their evaluation in EHW approaches.

generated by GP without a corresponding improvement in fitness. Additionally, Genetic Algorithms (GAs) also could be applied to evolve circuits, while some of them are designed without considering circuit validity and thus cannot be directly adapted for circuit evolution. Moreover, the incorporation of memristors into the design of novel EHW will also pose new challenges such as new representation design for the memristive circuits and variations existing in the memristive circuits. Limited by the current immature manufacturing process and the physical properties of the memristor device itself, various forms of errors will be appeared during the process of changing the states of memristor like reading error and writing error, etc. These errors will cause the actual memristor value to deviate from the target memristor value, resulting in a decline in the performance of the memristive circuit. Therefore, addressing these algorithmic challenges will be essential for advancing the field of EHW.

The aforementioned issues pose a demand for novel EHW designs based on memristors. Motivated by these, the focus of this thesis is to develop novel EHW based on memristors with improved circuit representations, more efficient evolutionary design processes, and broader applications of memristive circuits. This chapter is organized as follows. Section 1.1 discusses the research questions answered by the thesis. Section 1.2 summarizes the contributions and the organization of the thesis.

1.1 Research Questions

In light of the above motivations, we have formulated three research questions to guide our investigation into novel evolvable hardware based on memristor. The details and the methods for answering each research question will be presented in Chapters 2, 3, 4, and 5, respectively.

1.1.1 Improving the general applicability and Efficiency of the Circuit Evolution

Research Question 1 (RQ1): How to improve general applicability and efficiency of the circuit evolution?

Circuit evolutionary approach leverages evolutionary algorithms to optimize and evolve circuit designs, offering a promising avenue for novel EHW based on memristor. One of the key technologies of circuit evolution is how to code the circuit. Circuit coding methods directly affect the generalized applicability of the circuit evolutionary approach, so as to affect the performance and the efficiency of circuit evolutionary design. Generalized applicability refers to the ability of a circuit representation method to be applied across a wide range of circuit designs. A method with high generalized applicability can handle various types of circuits, including both of two-terminal devices and three-terminal devices, different circuit configurations, and different levels of complexity. It's not limited to specific types of circuits or components. In the context of circuit evolution, efficiency refers to the speed and computational resources required for the evolutionary process, including quick encoding/decoding, effective evolutionary operations, and the ability to discover novel circuit designs. However, existing circuit representations have certain limitations. For instance, the string-based representation can involve complex decoding processes, restricting the evolutionary process [190]. There are also some problems in existing tree-based circuit representations. It is inefficient for Koza's tree representation [148] to be mapped into the circuit netlists [78]. Chang et al. proposed a tree-based representation for synthesizing Resistor-Inductor-Capacitor (RLC) circuit [38], however, this representation has been criticized for lack of design ability [102] and can only be applied in the two-pole electrical elements [236]. The graph-based representation has the difficulties of taking suitable crossover operation to support the evolutionary process, leading only the point mutation applied [197]. Therefore, given the current research inadequacies in the circuit representation, it is valuable to investigate circuit representations

with improved generalized applicability and efficiency.

Furthermore, circuit evolution is the process of designing and constructing a circuit to provide a prescribed response to a specified excitation. Evolutionary algorithms employed in the domain of EHW design operate autonomously, disregarding both design rules and domain-specific expertise. Their primary objective is to enhance the automation process and minimize reliance on human effort during the synthesis procedure. There is a very large search space for the circuit evolution due to its large design space of circuit topology and circuit parameters [216, 36, 235]. As the primary evolutionary algorithm employed in hardware evolution, the evolutionary algorithm has a drawback in the form of inefficient evolution process as the scale of the problem grows. Randomly selecting genes in an individual for evolutionary operations may limit the search efficiency by discarding potentially useful subcircuits [112]. The presence of devices with zero contribution may also cause bloat, resulting in larger evolved circuits [112]. Therefore, given the current research inadequacies in the circuit evolution algorithm, it is valuable to research how to improve the efficiency of the evolutionary process for evolving circuits.

Motivated by the identified shortcomings in the generalized applicability and efficiency of circuit evolutionary approaches, Research Question 1 (RQ1) is formulated. To comprehensively address RQ1, we start by investigating the circuit representation phase, which is the initial step in circuit evolution. The investigation aims to uncover a novel circuit representation technique that exhibits increased generalized applicability and enhances the efficiency of evolutionary circuit design. To address RQ1 thoroughly, we need to study an evolutionary algorithm capable of enhancing the efficiency of the evolutionary process. Moreover, answering RQ1 will not only resolve current issues with circuit representation and the evolutionary process but also improve the overall generalized applicability and efficiency of the circuit evolutionary approach.

1.1.2 Evolution of Memristive Circuits Based on Reconfigurable Architectures

Research Question 2 (RQ2): How to evolve memristive circuits based on reconfigurable architectures?

RQ1 mainly focuses on improving the generalized applicability and the efficiency of the circuit evolutionary approach, which aims to solve some problems presented by existing approaches. However, RQ1 investigates an evolutionary approach for extrinsic EHW, which is to evolve the circuit design instead of the circuit evolution based on pre-designed reconfigurable architectures. Reconfigurable architectures could be used to better exploit the changeable, non-volatile, and other characteristics of the memristor, so that the computation and storage could be implemented synchronously, enabling the in-memory computing. They offer circuits with enhanced flexibility and adaptability, enabling them to fulfill various functions and accommodate changing conditions. The integration of memristive circuits within reconfigurable architectures has demonstrated significant potential across diverse applications, particularly in the field of brain-inspired computing, where they can emulate the behavior of biological synapses and neurons [333, 348]. Nevertheless, research on the evolution of memristive circuits within reconfigurable architectures remains limited. Hence, it holds great promise to explore the evolutionary aspects of memristive circuits within the framework of reconfigurable architectures, and it is valuable to investigate how to develop intrinsic EHW based on reconfigurable memristive architectures.

Moreover, memristors are a type of circuit element that exhibits resistance memory, meaning that their resistance varies based on the history of the applied voltage [281, 333]. This challenge is significant to memristive circuits in reconfigurable architectures and requires innovative approaches to overcome, since it not only exists in the memristor device, but also

the memristive arrays. Memristors are the dominant components of the memristor-based crossbar, such that the variations presented by the memristors can have a larger effect. For instance, in many cases, the memristor-based crossbar are used to map the weight matrix of neural networks. Therefore, if this crossbar cannot map the weights accurately due to variations presented by the memristors, that will impact the performance of the memristive neural network. Therefore, it is valuable to consider variations presented by memristors when developing intrinsic EHW approaches.

RQ2 focuses on the evolution of memristive circuits within reconfigurable architectures and addresses the issue of memristor variations. The evolution of memristive circuits based on reconfigurable architectures is investigated both from the hardware architectures and algorithm perspectives. The memristive reconfigurable architectures will be first constructed as the platforms for the circuit evolution. Based on the different memristive reconfigurable architectures, their corresponding circuit representation and evolutionary operators will be studied. By investigating the use of evolutionary techniques for the design of memristive circuits based on reconfigurable architectures, the memristive circuits could be evolved on-chip based on the reconfigurable architectures, further boosting the flexibility and adaptability of memristor, enabling them to fulfill various functions and accommodate changing conditions. This can lead to the development of more efficient and powerful brain-inspired computing systems, intelligent sensors, and other emerging technologies. Moreover, benefited by the evolutionary techniques, the negative influence of the memristor variations could be further prevented, and it is also very promising to harness these memristor variations transforming these negative influences into benefits.

1.1.3 Accurate and Energy-efficient EHW for Reservoir Computing

Research Question 3 (RQ3): How to improve the accuracy and energy efficiency of EHW for RC?

RQ2 mainly investigates novel evolutionary techniques, by which the memristive circuits can be developed using reconfigurable architectures. More complex applications and their resulting challenges need to be explored to further verify the effectiveness of the novel EHW based on memristors. RQ3 is proposed to scope further improvements of the accuracy and energy efficiency of EHW for RC applications. Since the properties of memristor (nonlinear behavior, fading memory) are well aligned with the key features of RC (nonlinear feature and short-term memory) and the evolvability of reservoirs is also the desired property to further improve its performance (regression or classification accuracy), we consider RC as a promising application of our proposed evolvable memristor-based EHW.

Reservoir Computing (RC) was originally proposed to provide solutions for the shortcomings of conventional Recurrent Neural Networks (RNNs). However, neuron-based RC may result in large area and power overhead caused by neuron connections especially from the hardware implementation perspective [8, 192]. Compared with these algorithms, Time Delay Reservoir (TDR) seems a good candidate for hardware implementation since it avoids this overhead by time multiplexing resources and utilizes a single neuron and a delayed feedback to create reservoirs. However, there are two existing issues for the TDR reflected in the algorithm and hardware aspects, respectively. In terms of the algorithm part, TDR has the limited ability of handling the long-range dependence time series for its short-term memory feature. This limited ability of handling the long-range dependency problem will degrade the accuracy of TDR for processing the time series. For the hardware part, current hardware implementations still use digital or partially digital components, which could lead to higher

power consumption.

Therefore, in order to solve these two existing issues, RQ4 is proposed to investigate accurate and energy-efficient EHW for reservoir computing. The answer to RQ4 will first analyze the memory property of standard TDR, which provide the theoretical foundation to the following research. Then, based on this theoretical analysis, we propose a memristive TDR with enhanced memory, which is more accurate when it handles the long-range dependence time series. Moreover, considering higher power consumption of digital or hybrid implementation to TDR, we further propose a full-analog implementation of TDR to answer the RQ4. This fully-analog TDR is more energy-efficient compared with digital or hybrid TDR.

1.2 Thesis Contribution and Organisation

In summary, the contributions of this thesis include:

1. We propose novel circuit evolution approach with improved design ability and efficient evolution process using genetic programming in Chapter 3 in answer to RQ1 of Section 1.1.1. Specifically, a novel circuit representation that can solve problems of existing approaches such as inefficient transformation to netlists and lack of generalized applicability is proposed. This novel circuit representation is based on a tree structure which can be more efficiently transformed into the circuit netlist and has greater generalized applicability for both of the two-terminal and three-terminal devices. This contribution has been published in our paper [*1].

2. Moreover, a circuit evolution algorithm based on the aforementioned tree representation is also proposed in Chapter 3 to improve the efficiency of the evolutionary processes

based on Shapley values, further complementing the answer to RQ1 of Section 1.1.1. Through Shapley values, the contribution of each function node can be identified in a circuit-plausible way. Based on this circuit-plausible importance to the functional nodes, a two-stage evolution framework for evolving circuit is further proposed to enhance the efficiency of the evolutionary process. This contribution has been published in our paper [*2].

3. We propose an indirect circuit representation for 4T1R memristive reconfigurable architecture and its corresponding evolutionary operators for the memristive circuit evolution, which are more suitable for the reconfigurable memristive architecture. The proposed approaches achieve the on-chip circuit evolution based on memristive reconfigurable architectures, by which the flexibility and adaptability of the proposed EHW can be enabled. Moreover, the negative impacts of the memristor variations could be prevented effectively by the proposed indirect circuit representations. These are the answers to RQ2 of Section 1.1.2 and will be presented in Chapter 4. This contribution has been published in our papers [*3].

4. Moreover, we also propose an indirect circuit representation and its corresponding evolutionary operators for 1R memristive reconfigurable architecture, further complementing the answer to RQ2 of Section 1.1.2. 1R is the passive memristive array that doesn't have the transistor in the array, which is more compact in hardware implementation compared with 4T1R memristive array. However, there will be the sneak current issue in the 1R memristive array, while 4T1R array can prevent it by the transistors. Instead of regarding the variations from the 1R memristive array as the negative influences, we further harness these variations in the 1R memristive array with our proposed evolutionary technique, where the nonlinearity and fading memory of the sneak currents are harnessed to implement the memristive reservoir circuit with a more compact architecture. These are the answers to RQ2 of Section 1.1.2 and will

be presented in Chapter 4. This contribution has been included in our paper [**1].

5. We develop a high-order time delay unit for TDR and its memristive implementations, which can store and transfer long-term history states, thus enhancing the accuracy and energy efficiency of TDR. Specifically:

   (a) In terms of the algorithm perspective, we first propose a high-order time delay unit to capture more complex temporal patterns and dynamics in the reservoir states, which enhances the memory of TDR and improves the performance of the reservoir computing system. The order of the time delay units can be optimized to adjust to different tasks, implementing the adaptive memory enhancement.

   (b) In terms of the hardware perspective, we also propose a memristive implementation of our proposed adaptive memory-enhanced TDR using two different types of memristors, which benefits from the energy efficiency of the memristors.

   These are the answers to RQ3 of Section 1.1.3 and will be presented in Chapter 5. This contribution has been published in our paper [*4].

6. Additionally, we work introduces a new TDR with improved memory and its fully analog memristive circuit design, which can further enhance the power efficiency of the hardware implementation.

   (a) In terms of the algorithm perspectives, we propose a novel TDR with dual memory enhancements that can capture both short-term and long-term dependencies in the input time series. The first part of our model leverages a novel dynamic input masking technique that utilizes the historical information of the input to capture the short-term dependencies. The second part of our model handles the long-term dependencies by using the reservoir state forwarding technique to strengthen the interaction and accumulation of long-term states.

(b) In terms of the hardware perspective, to further boost the energy efficiency of proposed novel TDR, we design a fully analog hardware system, which consists of the memristive implementation of the input, reservoir, and output layers. A dynamic memristor is used as a nonlinear node and nonvolatile memristor is used to build memristor-based window element and memristor-based delay element.

These are the answers to RQ3 of Section 1.1.3, and its contribution has been included in our paper [**2].

The work resulting from our investigations has been reported in the following publications:

Refereed published papers:

[*1] **Xinming Shi**, Leandro L. Minku, and Xin Yao, "A Novel Tree-based Representation for Evolving Analog Circuits and Its Application to Memristor-based Pulse Generation Circuit," *Genetic Programming and Evolvable Machines*, 23, 453–493 (2022). https://doi.org/10.1007/s10710-022-09436-w (This paper supports Chapter 3)

[*2] **Xinming Shi**, Jiashi Gao, Leandro L. Minku, and Xin Yao, "Evolving Parsimonious Circuits Through Shapley Value-based Genetic Programming," *in Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Boston, USA, 2022, pp. 602–605 (This paper supports Chapter 3)

[*3] **Xingming Shi**, Leandro L. Minku and Xin Yao, "Evolving Memristive Reservoir," in *IEEE Transactions on Neural Networks and Learning Systems*, doi: 10.1109/TNNLS.2023.3270224. (This paper supports Chapter 4)

[*4] **Xingming Shi**, Leandro L. Minku and Xin Yao, "Adaptive Memory-Enhanced Time Delay Reservoir and its Memristive Implementation," in *IEEE Transactions on Com-*

puters, vol. 71, no. 11, pp. 2766-2777, 1 Nov. 2022, doi: 10.1109/TC.2022.3173151. (This paper supports Chapter 5)

Papers under review:

[**1] **Xingming Shi**, Leandro L. Minku and Xin Yao, "Harnessing Sneak Currents in 1R Memristor-based Crossbar by Spatiotemporal Evolution for Reservoir Computing," in *IEEE Transactions on Neural Networks and Learning Systems*, 2022. (This paper supports Chapter 4)

[**2] **Xingming Shi**, Leandro L. Minku and Xin Yao, "Time Delay Reservoir with Enhanced Memory and Its Fully Analog Memristive Circuit Design," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023. (This paper supports Chapter 5)

Published papers inspired by the new research directions proposed by this thesis:

[***1] **Xinming Shi**, Jiashi Gao, Leandro L. Minku, James Jian Qiao Yu and Xin Yao, "Second-order Time Delay Reservoir Computing for Nonlinear Time Series Problems," *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, Orlando, FL, USA, 2021, pp. 1-8, doi: 10.1109/SSCI50451.2021.9659913.

[***2] **Xinming Shi**, Zilu Wang, Leandro L. Minku, and Xin Yao, "Explaining Memristive Reservoir Computing Through Evolving Feature Attribution," *in Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO)*, Lisbon, Portugal, 2023, doi: 10.1145/3583133.3590619.

[***3] Zilu Wang, **Xinming Shi**, and Xin Yao, "A Brain-Inspired Hardware Architecture for Evolutionary Algorithms based on Memristive Arrays," *ACM Transactions on Design Automation of Electronic Systems*, 2023, doi: 10.1145/3598421.

Chapter Two

Background and Literature Review

This chapter gives the background knowledge and reviews the literature relevant to this thesis. Section 2.1 reviews the hardware platforms to EHW based on the traditional electronic devices and emerging electronic devices, motivating the investigation of novel EHW based on memristors from the hardware perspective. Section 2.2 reviews the different types of circuit representation and circuit evolution algorithms, from which RQ1 is motivated though existing issues of limited design ability and evolutionary efficiency and RQ2 is identified to investigate how to evolve the reconfigurable memristive architecture with indirect circuit representation. Section 2.3 reviews the reservoir computing applications of EHW, from which RQ3 is motivated though the existing issues of handling the long-term dependency problem and the higher power consumption attributed by the digital or hybrid circuit implementation. Section 2.4 introduces the benchmark circuits applied in this thesis to verify our proposed approaches. Section 2.5 summarizes this chapter.

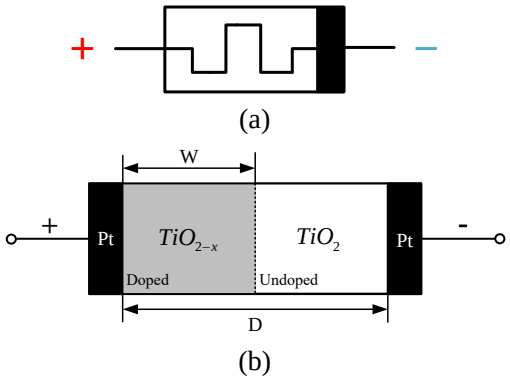


Figure 2.1: (a) The schematic representation of a memristor; (b) Physical model of HP memristor

2.1 Hardware Platforms for EHW

The key elements in EHW are the Evolutionary Algorithms (EAs) used and the programmable devices over which the circuits are deployed. Therefore, researching the different platforms on which evolved circuits can be evaluated is important for this work. Since around 1950s, evolutionary characteristics has been implemented into hardware [239]. More recently, with the rapid development of microelectronics, the size of electronic components has decreased from 2-3 μm in 2002 to 0.09 μm, which allows for the faster development of implementing EHW and greater advancement in the field of EHW [304].

However, the speed and energy efficiency of these traditional device-based hardware platforms are restricted due to the bottleneck of the von Neumann architecture and the Moore's Law is reaching its limits [166]. As a potential replacement for traditional device in hardware circuit design, memristors can provide several advantageous features such as non-volatility, high density, low power, and good scalability [134]. The related work to memristors will be introduced in Section 2.1.1. Therefore, the related work of hardware platforms to EHW based on conventional electronic devices and memristors will be introduced in Sections 2.1.2 and 2.1.3 , respectively.

2.1.1 Memristor

From the perspectives of symmetry and electromagnetic field, Chua first predicted the existence of the fourth fundamental circuit element besides resistor, capacitor, and inductor, which is called memristor that can represent the relationship between the change of magnetic flux and the change of electric charge [48]. In 2008, R. S. Williams' team at HP Labs fabricated a physical memristor with typical characteristics and published an article titled "The missing memristor found" in *Nature*, announcing the discovery of memristor [281]. Since HP team fabricated memristor, memristor has been widely used as a synapse in neural networks due to its nanoscale size, resistance change with external voltage, and non-volatility characteristics, which meet the requirements of circuit implementation of artificial neural networks. The schematic representation of a memristor is given in Figure 2.1 (a). This type of memristor is based on TiO_2 solid-state thin film memristor, whose structure uses metal Pt as the two electrodes, and the resistance switching layer is partially divided into doped and undoped regions, which are composed of TiO_2 and TiO_{2-x} respectively (shown in Figure 2.1 (b)) . Compared with TiO_2, TiO_{2-x} has less oxygen ions, which produces oxygen vacancies with +2 valence charge. For this type of semiconductor material, the oxygen deficiency sensitivity makes the doped region have high conductivity under the applied positive voltage. At the same time, with the action of the positive electric field, the right boundary of the doped region moves to the undoped region, making the width of the doped region gradually increase, further reducing the resistance and increasing the conductivity. The device shows a conductive (ON) state. On the contrary, under the applied reverse voltage, the left boundary of the undoped region moves to the left under the reverse electric field, reducing the width of the doped region, resulting in a decrease in conductivity and an increase in resistance. The device shows a disconnected (OFF) state.

Its application has changed the pattern of traditional electronic devices. Therefore, it

has received increasing attention from the electronics community and has been researched in various aspects. In terms of memristor fabrication, memristors based on different materials have been proposed, such as spintronic memristor [118], ferroelectric memristor [39], and drift memristor [360], etc. In terms of memristor application, due to the following characteristics of memristor:

(1) Compared with traditional devices, memristor has better computation density, which allows for more efficient data processing, potentially leading to faster computational speeds and more compact device architectures;

(2) Memristor is an analog device that can realize continuous output, which is suitable for representing continuous changing weights;

(3) Memristor is a passive device and has non-volatility after power-off, so it has low power consumption characteristics;

(4) Because the resistance of memristor changes with the change of magnetic flux and charge, it has memory and learning abilities that are required for neural network synapses;

(5) By constructing a memristor crossbar neural network, the storage capacity and information processing capacity of the neural network can be better improved. Therefore, it is often used as a synapse in neural networks, and its weights are trained to further realize the function of neural networks.

Therefore, in light of these aforementioned advantages, memristiors have been widely applied to construct the brain-inspired computing systems [333, 345]. Traditional computing systems are mainly based on Complementary Metal Oxide Semiconductor (CMOS) transistors, which have limitations in storage capacity and transmission efficiency, resulting in high energy consumption and computational burden [117, 169, 289]. Hence, there is a need

to explore beyond-CMOS devices to build brain-inspired computing systems with higher performance [122]. Memristors are emerging two-terminal memory devices that have many appealing features as aforementioned [134, 301, 321]. They can emulate neurons or synapses in brain-inspired hardware systems [170, 265, 222], and can be arranged into crossbar array architectures that enable parallel in-memory computing and high-density storage [226]. For example, the memristive crossbar array can execute multiply-add operations in parallel in the analog domain, which can facilitate the acceleration of some computing-intensive modules in artificial neural networks with high throughput and low energy and area consumption [43, 321]. Despite the potential of memristors, their practical implementation and application are currently hindered by challenges such as variation and reliability issues. The primary concerns stem from device variation and reliability issues, which can be attributed to the still-developing fabrication processes and the physical limitations of the devices. These challenges can lead to inconsistent behavior among memristors, making it difficult to design systems that rely on uniform performance [335]. Moreover, the immature state of memristor technology means that there is a lack of standardized models for accurately predicting their behavior, which is crucial for simulation and design purposes. This unpredictability can hinder the integration of memristors into existing hardware systems, requiring additional research and development to ensure compatibility and functionality [3]. Another significant drawback is the endurance of memristors, as they can degrade over time with repeated use. This degradation can affect the stability of the memristive states, leading to data retention issues and a decrease in the overall lifespan of the device [335]. Despite these drawbacks, the potential benefits of memristors such as their high density, low power consumption, and the ability to perform in-memory computations, remaining compelling reasons for their continued exploration in EHW [333]. Researchers are actively working to overcome these challenges [332, 120, 336]. The ongoing advancements in memristor technology are crucial for realizing their full potential in creating efficient, adaptive, and intelligent EHW systems.

In summary, compared with traditional devices, memristor has the advantages of nanoscale size, memory and learning ability, reconfigurability, continuous output, good compatibility, etc., which meet the needs of investigating the novel EHW. Our aim is to explore the feasibility of designing a novel EHW system using memristors, leveraging their variations to implement diverse functions within this memristor-based EHW system.

2.1.2 Hardware Platforms for EHW based on Conventional Electronic Devices

In 1993, the concept of evolutionary hardware (EHW) was introduced by researcher Higuchi et al. [109]. This led to a worldwide research boom and the establishment of institutions for EHW research in many countries. Initially, Field Programmable Gate Array (FPGA), Field Programmable Analog Arrays (FPAA), Field Programmable Transistor Arrays (FPTA) and analog hardware evolution circuits based on analog functional blocks such as transistors, resistors, and wires were used as platforms for implementing hardware evolution [347]. Some EHW were implemented by FPTAs and FPAAs. Keymeulen et al. proposed a fault-tolerant EHW using FPTA [140] for evolving a fault-tolerant digital circuit (XNOR), and Santini et al. proposed an evolutionary analog circuit design on FPAA for evolving an operational amplifier, a logarithmic amplifier and a membership function circuit of a fuzzy logic controller [244]. However, due to the highly flexible programmability of FPGAs, they became the mainstream platform for implementing evolved hardware. This was demonstrated by Garis and Korkin, who successfully developed a bionic brain evolution hardware system with a multi-chip FPGA cascade [57]. The majority of available research results on EHW have been implemented on FPGAs, with significant contributions from Sekanina. Sekanina et al. [252] proposed an evolutionary circuit structure for implementing Virtual Reconfigurable Circuit (VRC) on FPGAs, solving the difficulties caused by the FPGA configuration bitstream

at that time. Based on research from Sekanina, various application research results have been proposed using FPGA to implement EHW. For example, Salvador et al. proposed the implementation of EHW-based adaptive image filters [242], Glette et al. proposed the design of a sonar spectrum classifier based on an online evolved hardware system [86], and Wang et al. proposed the evolution of combination logic circuits on a VRC-based evolved hardware system [315]. As evolved hardware research advanced, it became a national strategy in the US, with the ESA and NASA setting research goals of applying EHW technology to aerospace electronic systems to improve system reliability [76, 200].

2.1.3 Hardware Platforms for EHW based on Memristors

With the rapid development of microelectronics and semiconductor technology, scaling-driven performance improvements within the framework of traditional analogue and digital design become progressively more restricted by fundamental physical constraints [256]. Therefore, the need for reconfigurable hardware operating close to the fundamental limits of energy consumption became increasingly pressing. Emerging nanoelectronics technologies brought forth a new prospect, where memristors have been widely used as reconfigurable analog blocks in crossbar arrays over the past half a decade. The ability of memristors to act as thresholded electrically tuneable [96], multilevel [275], non-volatile resistive loads [324], combined with their inherently scaling-friendly [334] and low power consumption [299] has rendered them a highly promising candidate for use in future electronics applications. These properties promote memristors as ideal candidates for achieving in silicon reconfigurability in a post-Moore context [256]. Figure. 2.2 shows two examples of the memristive reconfigurable architectures, which are 4T1R architecture [338] (Figure. 2.2 (a)) and 1R architecture [333] (Figure. 2.2 (b)), respectively. In 4T1R memristor-based reconfigurable architecture, the memristive states can be reconfigured by applying different control signals

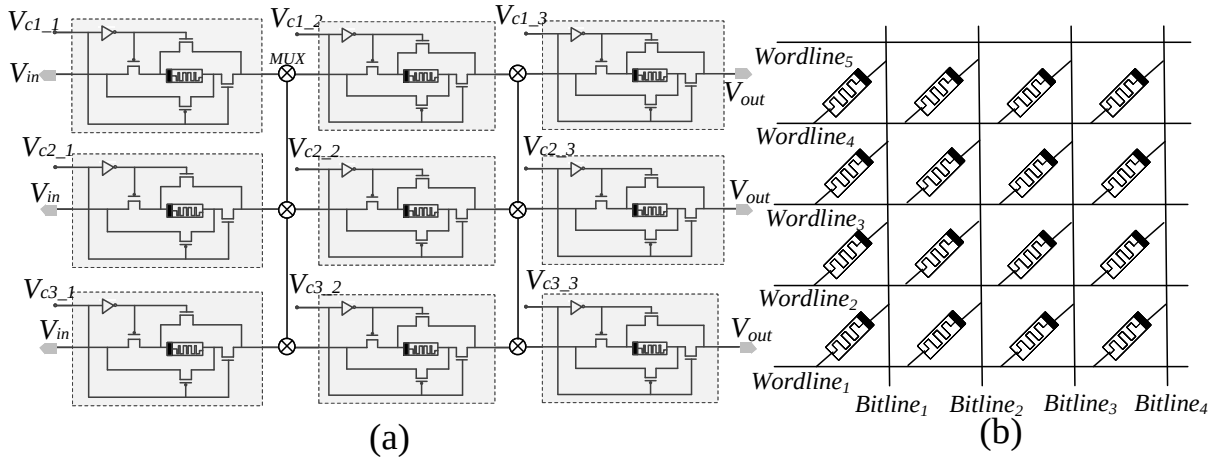


Figure 2.2: Two examples of the memristor-based reconfigurable architectures. 4T1R architecture (a), 1R architecture (b). In 4T1R architecture, V_c denotes the applied control voltage, V_{in} and V_{out} refers to the input voltage and output voltage, respectively. In 1R architecture, the memristor is connected between the bitline and wordline, respectively.

V_c to the crossbars, and the memristive circuit topology can also be reconfigured by combining with multiplexing devices. V_{in} represents the input signals to the 4T1R memristor-based reconfigurable architecture, and V_{out} denotes the output voltage of the architecture. In 1R memristor-based reconfigurable architecture, the memristor is connected between the bitline and wordline.

Robinson et al. [227] proposed a self-reconfigurable analog resonant computer, which employed a fixed electronic circuit schematic performing computing logic operations (for example OR, AND, NOR, and XOR Boolean logic). This self-reconfigurable unit is composed of resistors, capacitors, inductors, and memristor devices. During the computational logic self-reconfiguration process, training input signals were applied to alter the impedance state of the memristor device. Once the training process was completed, the circuit was probed to verify whether the desired logic operation had been programmed. Although memristors have been applied in this self-reconfigurable unit, the capacitors and inductors of units occupy large area of circuit, which may lead to a bulk auxiliary circuit. Moreover, the proposed self-reconfigurable unit was designed solely for digital purposes, focusing on logic operations,

and no attempt was made to apply it in analog applications.

Kejie et al. [119] proposed a diode-memristor-based bit cell 1D2R. This structure consists of two memristors and a diode, used to simultaneously program the memristors. The 1D2R cell was to replace both the Static Random Access Memory (SRAM) cell and routing switch in the FPGA. Besides the routing resources, logic blocks which mainly consist of LUTs and Flip-Flops have also been renovated. However, this 1D2R unit still has some drawbacks in terms of circuit design and application. First, one diode is applied to the circuit, while the size of diode is much larger than that of memristors, which may offset the superiority of circuit size offered by the memristors. Second, the application of the proposed 1D2R unit is still limited to the digital applications.

Kumar et al. [157] proposed a new memristor-based two-input LUT that employed a control circuit to select distributed memristors. However, a large auxiliary circuit is necessary for the only two-input LUT, which may increase the burden of circuit size and power consumption of the whole architecture. Seungbum et al.[15] proposed a reconfigurable multiplier architecture based on memristor-CMOS to reduce the area occupation and increase flexibility for various applications according to the size of input data multiplier and multiplicand. However, this work still has the problem of limited application to the digital area. Serb et al. [256] proposed the seamlessly fused digital-analogue reconfigurable computing using memristors, which lays the conceptual foundations and provides experimental proof of a design approach for electronics formed by marrying standard.

Le et al. [338] proposed a 4T1R memristive unit, which consists of four transistors and one memristor. The 4T1R memristive unit, shown in Figure 2.2 (a), uses a control signal V_c to control the current direction across the memristor. This flexible control and its controllable dynamical behaviors enable its successful application to the associative memory. The 1R memristive architecture, shown in Figure 2.2 (b), is a passive array that has no transistors

in the memristor crossbar. It is attractive for its low power consumption and high packing density potential, but it suffers from sneak path current and half-select issues that hinder the proper programming or reading of memristors in a large passive array. The half-select issue arises when unselected memristors in the same row or column as the selected one also experience a fraction of the applied voltage, leading to incorrect data being written or read and resulting in high energy consumption [333].

To conclude, memristor-based reconfigurable systems have shown remarkable advantages over conventional CMOS technologies in terms of power consumption, scalability, and adaptability [333, 256]. However, to the best of our knowledge, no research has explored the use of memristors in the field of EHW for analog intelligent applications, such as time series prediction, speech recognition, and nonlinear system identification. It is worthwhile to combine the memristive reconfigurable architecture with EHW for intelligent computing systems, so that the intelligent systems can be not only equipped with the evolvability benefited by the evolutionary approach, but also with the compact size and energy efficiency advantaged by the memristors.

2.2 Circuit Representations and Circuit Evolution Algorithms

Besides the hardware platform to EHW, the algorithms play an important role in developing EHW. The complex topology and a large number of component values for circuits makes the circuit evolution a challenge. Circuit representation is the first consideration in automated circuit design. It indicates how to encode a circuit. According to the taxonomy of the circuit representation proposed in [302], the circuit representations could be divided into the direct circuit representation and indirect circuit representation aimed to encode different types of

the EHW, which will be introduced in Sections 2.2.1 and 2.2.2. Moreover, the evolutionary algorithm is also indispensable to design EHW, where the related work to the evolutionary algorithm of EHW will be introduced in Section 2.2.3.

2.2.1 Direct Circuit Representation

Direct circuit representation is a method of representing a circuit directly corresponding to the circuit's components and connections, which are normally used to evolve the extrinsic EHW [302]. This method provides a detailed view of the physical layout of the circuit, including the specific devices used and how they are connected. This approach is more straightforward and can be more intuitive for those with a background in electronics. We also use the term "direct circuit representation" to refer to a method of representing a circuit directly corresponding to the circuit's components and connections. There have been several studies on the direct circuit representation as it is an intuitive way to represent the circuits. Mattiussi et al. proposed Analog Genetic Encoding (AGE) based on string representation to synthesize analog circuits [190]. The AGE genome is composed of string of characters, which characters denote different meanings such as device names and parameters. Each gene of AGE genome denotes a device, in which two regions are included for denoting each terminal and parameters of the device. Connection of these devices is determined by device interaction map. The final circuit is constructed by connecting all devices step by step according to device interaction map. The experimental results verify that AGE can synthesize analog electrical circuits. However, this representation is so complex that much computation time is taken during decoding process [78]. The direct circuit representation method encodes a circuit according to a schematic that details the devices and their interconnections [294, 252]. This method provides a detailed view of the physical layout of the circuit, including the specific devices used and how they are connected. This approach is more straightforward

and can be more intuitive for those with a background in electronics.

Researchers also have attempted to apply graph structure to represent circuits directly [78, 194]. Graphs are attractive representations for circuits as they are more intuitive than the usual tree representation [197]. An Evolutionary Graph Generation (EGG) system was proposed to synthesise digital circuits [41] [7]. This system is also applied into synthesis of analog circuit [213]. In EGG, a node is used to represent a device or I/O pin, and an edge is used to represent connection between devices and I/O pins. Some evolutionary operators are designed to modify individuals through directly changing the graph structure. However, the graph-based representation has the difficulty of not having a suitable crossover operation, leading only to the point mutation being applied [197].

2.2.2 Indirect Circuit Representation

The indirect circuit representation does not explicitly encode the circuit topology and devices. Instead, it encodes a circuit based on configuration signals or other descriptive elements [302]. This can provide a higher level of abstraction, allowing for more flexibility in the design and optimization process [148]. This is in contrast to the direct circuit representation method explained in Section 2.2.1, which encodes a circuit according to a schematic that details the devices and their interconnections.

However, it should be noted that the term "indirect circuit representation" may have different meanings in different contexts. For example, some researchers may consider indirect representation to be a method of encoding the circuit structure and functionality in a compact and abstract way, without specifying the low-level details of the hardware implementation. In this sense, indirect representation is contrasted with direct representation, which is a method of encoding the circuit configuration bitstream that directly corresponds to the

hardware components and connections [302]. This definition of indirect representation is more general and encompasses various types of mappings from genotype to phenotype, such as grammatical evolution, cartesian genetic programming, and developmental systems. In this thesis, we adopt the definition of indirect representation as a method of encoding the circuit configuration signals as different circuit. We believe that this definition is more suitable for our research problem, as we aim to evolve EHW that can adapt to changing environments and tasks.

Tree-based circuit representation has been regarded as a crucial factor allowing the greater diversity among evolved topology [236] [38]. Koza and his collaborators have done lots of research on automatic synthesis of analog circuits by means of Genetic Programming (GP) [149] [23], where the tree-based circuit representations were applied and exhibited good diversity and exploration ability [37]. However, there are still some problems in existing Koza's tree-based circuit representations. First, the size of the evolved design may be large within the tree representation [54]. Second, it is too complex for the tree representation to be mapped into the circuit netlists [78]. In 2006, Chang et al. proposed a novel tree-based representation for synthesizing RLC circuit [38], which is more efficient for passive circuit synthesis. However, this representation has been criticized for lack of design ability [102] and can only be applied in the two-pole electrical elements [236].

Some researchers encode the circuits as an expression whose syntax can be described by a grammar. Developmental expressions are often used in this approach, which define a sequence of specialized functions that transform an initial circuit, known as an embryo circuit, into a final circuit [150]. An alternative approach to circuit encoding involves the use of grammars, which can be implemented through grammatical evolution. In this method, linear binary chromosomes are utilized as coding structures, and a specific set of sub-circuit development expressions, as defined by a given grammar, are employed to decode each chromosome into a circuit. This technique provides a flexible and powerful method for encoding

circuits that can accommodate a wide range of sub-circuit types and configurations. Furthermore, the use of grammars enables the generation of circuits that are syntactically and semantically correct, which can improve the efficiency and effectiveness of circuit design and optimization [36].

Drawing inspiration from biological Genetic Regulatory Networks (GRN), some researchers [189, 190] have explored the use of encoding implicit interactions between components in circuit design. This approach involves encoding both the components and their connections with other components into a linear chromosome. The genes in this chromosome contain regulatory regions that can be influenced by other gene coding regions. The interactions between these regulatory regions determine the actual influence and final weight of the connections between the components. This method enables the design of circuits that can exhibit complex emergent behaviors, similar to those observed in biological systems.

Moreover, after inducing the memristors to EHW, the circuit representation may need changes. Different indirect circuit representation of memristor need to be designed for different reconfigurable architectures. There are various types of memristive reconfigurable architectures, therefore, it is valuable to investigate indirect circuit representations for them, to potentially further improve the design ability of EHW based on memristors.

2.2.3 Evolutionary Algorithms for Evolving Circuits

Researchers proposed several ways to tackle the challenge of implementing automated analog circuit design, including domain knowledge-based and Evolutionary Algorithm (EA)-based approaches [270, 300]. Table 2.1 gives a brief comparison of automated circuit design methodologies based on domain knowledge and EA.

As the name implies, knowledge-based methods rely on the knowledge obtained from

Table 2.1: Characteristics of automated circuit design methodologies based on domain knowledge and EA [270].

Based Methods	Structures	Complexity	Knowledge required
Knowledge	Familiar	Very high	Very high
EA	Unfamiliar	Low	Low

specific circuits or sub-circuits [98], thus the great human effort and experience is highly required to extract the knowledge for each generated structure one by one [270], which is difficult especially to inexperienced developers. In addition, the computing complexity of some knowledge-based methods [146] is high, as it needs to check if the automatically generated part is potentially useful by circuit knowledge such as transmission parameters, incurring high computing complexity.

EAs applied to EHW design make the synthesis process more independent of human circuit design effort [343]. Given an EA created to evolve hardware, its application to evolve a new piece of hardware does not require further human circuit design effort. In addition, EAs for EHW can explore a much wider range of design alternatives [343]. Therefore, the EA-based methods involve less complexity, even though the structure of evolved circuit is unfamiliar compared with that of knowledge-based methods. Based on various EAs, different works of automatic circuit design have been proposed. Genetic Algorithm (GA) has been widely applied in the automatic circuit design. For example, in 1995, researchers have proposed a prototype synthesis tool DARWIN based on GA to design the CMOS opamp [155]. Aimed to synthesizing RLC circuit, a circuit synthesis framework was presented based on GA in 2006 [38]. Grammatical Evolution (GE), initially introduced by O'Neill et al. [216] and later applied by Castejón et al. [37], has proven to be a highly effective approach for the automatic design of analog circuits. This algorithm is based on grammar and can generate code in different programming languages using variable-length linear binary strings. Such versatility and adaptability make GE a powerful tool for tackling the challenges of analog

circuit design with success.

Federico et al. [37] manipulated GE to generate analog circuits automatically, and showed evidence about how GE can be used successfully for the automatic design of analog circuits. GP proposed by Koza John, made significant advancement on automatic analog circuit design [149] [23]. Some researchers commented that GP makes it possible for generating topology of analog circuits with arbitrary connections [236]. Several earlier researchers proposed that GP is likely the most successful evolutionary computation-based approach for analog circuit synthesis [178]. Even in recent years, GP (and its variants) is still considered as the evolutionary paradigm with the successful results in the field of analog circuit design as its ability to explore the design space [37]. However, crossover and mutation operators are typically used to create offspring in standard GP. Randomly selecting genes for such evolutionary operations may limit search efficiency by discarding potentially useful sub-circuits [112]. Such random nature of the evolutionary operations may also limit the efficiency of the evolution process by damaging or missing the important circuit blocks for better fitness. The presence of devices with zero contribution may also cause bloat, resulting in larger evolved circuits [112]. These constrains the development of GP, where the researchers make attempt to incorporate more advanced informed approaches with semantic information to improve the capability of exploring the search space [74, 206, 224].

In summary, related work on circuit representation and circuit evolution algorithms has some limitations, such as the limited design ability and inefficient evolutionary process, motivating RQ1, that is *How to improve general applicability and efficiency of the circuit evolution?*. Moreover, after inducing the emerging electronic device, memristor, to novel EHW design, the circuit representation may need changes. There are various types of memristive reconfigurable architectures, which may require different circuit representations to capture their features and behaviors. Therefore, it is valuable to investigate the indirect circuit representations for them, as indirect representations are more flexible to accommodate different

types of signal to evolve various reconfigurable architectures. Indirect representations are also usually more compact. This motivates RQ2, that is *How to evolve memristive circuits based on reconfigurable architectures?*

2.3 Evolving Reservoir Computing

Inspired by natural evolution and neuroscience, some researchers showed that the original RC paradigm uses fixed randomly created reservoirs is not evolutionary stable, thus researching evolutionary reservoirs has been regarded as the natural idea [255, 342]. And there are now several studies altering reservoirs to purse better performance on a given application [248]. Moreover, as mentioned in [17] and [180], most of the reservoir computing optimization and improvements are in the dynamic reservoir itself. It also confirmed the notion that the construction of a random fixed reservoir is not "good" enough. In [47], both of the alteration of node number, reservoir connectivity and weights have been studied, where the connectivity as well as the input scaling play a big role in the studied method, the quantities (number of connections) as well as the quality (values) of weights affect highly the training proceeding. However, the large searching space caused by the vast number of connection combinations and scalable weights makes it difficult to manually design a capable structure for a given task, and a considerable amount of domain knowledge may also be required. Therefore, there are existing issues of standard reservoir computing, such as handling long-term dependency problem and the higher power consumption attributed by the digital or hybrid circuit implementation. Evolutionary approach and memristive architectures may provide the possible solutions to these issue.

Therefore, reservoir computing is a very promising application field for the novel EHW based on memristors , by which the optimal reservoir design could be evolved on

the EHW. It motivates the RQ3 of the thesis, which is *How to improve the accuracy and energy efficiency of EHW for RC applications?* Section 2.3.1 reviews the reservoir computing models, Section 2.3.2 reviews the hardware implementation of the reservoir computing, and Section 2.3.3 reviews the evolution of the reservoir computing.

2.3.1 Reservoir Computing Models

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 [307, 42], and constitute one of the core paradigms of RNN modelling.

The reservoir is typically created as a recurrent neural network whose weights are randomly set and remain unchanged during training time [128]. The nonlinear transformation performed by the reservoir allows the original features, often in time domain, to be mapped as features in the reservoir states, which can then be further processed by a small, trained linear neural network, termed the readout layer. To reflect time correlations of input signals in the output signals, the reservoir needs to generate history-dependent dynamics. Another key feature of the reservoir is the fading memory, i.e. short-term memory property, which means that the reservoir state depends not only on the present inputs but also on inputs from the recent past (but not the far-past).

There are two different models of reservoirs to implement these aforementioned features. Figure 2.3 presents two RC models, which is using neurons and their connections to stimulate the features of the reservoirs (shown in Figure 2.3 (a)) and also using the nonlinear

system with delayed feedback to stimulate the features of reservoirs (shown in Figure 2.3 (b)). Generally, the reservoir computing consists the input layer, the reservoir layer and output layer. The input layer is responsible for feeding the input signal to the reservoir layer. The input signal can be any time-series data, such as speech, text, or images. The input layer can also perform some preprocessing or encoding of the input signal, such as scaling, normalization, or feature extraction. The reservoir layer is the core of the reservoir computing framework, which could be typically either random connections (shown in Figure 2.3 (a)) or one nonlinear system with several possible delay feedbacks, resulting in several virtual nodes (shown in Figure 2.3 (b)). The output layer is responsible for reading the reservoir states and mapping them to the desired output. The output layer is usually a simple linear readout mechanism that can be trained by a regularized linear least-squares optimization procedure.

As for the first model shown in Figure 2.3 (a) it includes several different variants such as Echo State Network (ESN) [128] and Liquid State Machine (LSM) [184], showing excellent performance in various nonlinear time series problems, such as speech recognition [269] and nonlinear system prediction [129]. They have been used to solve many computational problems, such as temporal pattern recognition, prediction, and generation tasks [129, 266]. Both of ESN and LSM are originally composed by the fixed randomly-generated reservoir architectures and neuron weights, where a reservoir with H neurons will have up to H^2 connections, potentially incurring a large area and power overhead of a reservoir [192]. Moreover, RCs with randomly-generated topology are also complex to be implemented in hardware, where the routing complexity, area, and power consumption are high [62]. As indicated in Figure. 2.4, the Time Delay Reservoir (TDR) implementation demonstrates superior scalability compared to the Echo State Network (ESN) implementation. In the TDR implementation, the number of Mosfets does not increase with the number of neurons. Conversely, in the ESN implementation, the number of Mosfets increases proportionally to

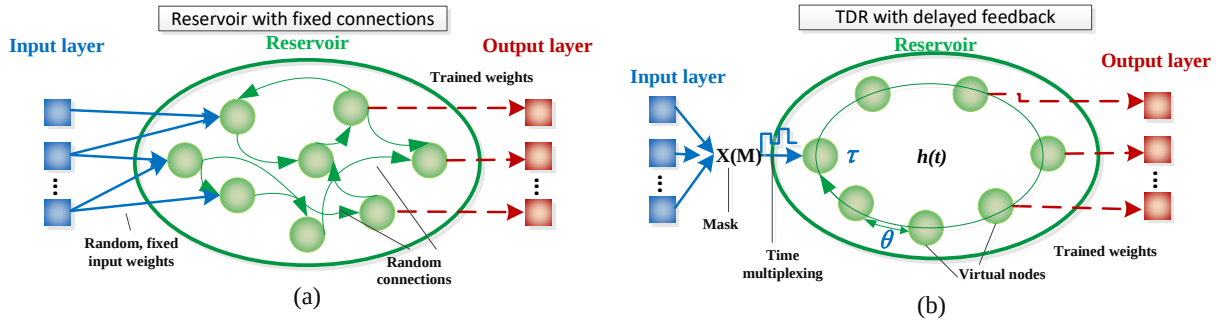


Figure 2.3: Two reservoir models: (a) reservoir with random connections (b) TDR with delayed feedback.

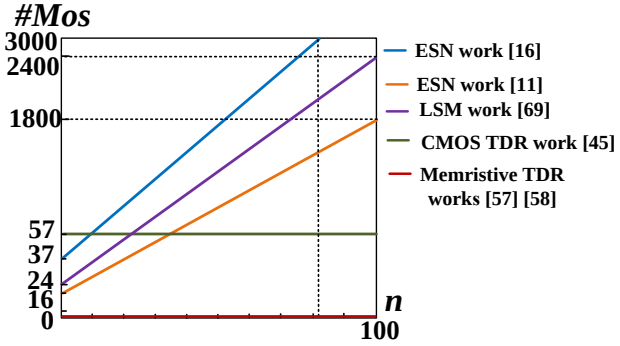


Figure 2.4: The relationship between the number of neuron and the number of Mosfets.

the product of the number of Mosfets in one neuron and the total number of neurons. This difference highlights the more efficient use of resources in the TDR implementation.

In fact the reservoirs do not necessarily need to be neural networks [180]. Some researchers have applied differential equation-based dynamic systems to model the high dimensionality of the reservoir instead of neurons [61]. The generality of differential equations provides the great potential of various dynamic systems to implement reservoirs [61]. Figure 2.3 (b) shows another type of reservoir using delayed feedback of nonlinear system to stimulate the features of reservoirs [8], which can generate a large number of virtual neurons in the neural network by time-multiplexing the input signal along the delay line. This can prevent the large overhead of neuron-based RCs such as ESN by using time multiplexing resources [192]. Therefore, TDR has friendly features to physical implementations. However, TDR performs poorly on tasks that involve long-term dependency [258, 264].

Moreover, the exploration of unconventional substrates has led to innovative approaches that extend beyond traditional materials. Julian Miller has advanced the use of liquid crystals, leveraging their unique electro-optical properties to function as adaptable reservoirs [97]. Carbon nanotubes, explored by Susan Stepney and Martin Trefzer, offer a highly conductive yet flexible platform, capitalizing on their molecular structure for enhanced computing efficiency [52]. Richard Evans' work with magnetic thin films utilizes the magnetic properties to store and process information in a compact form [51]. These substrates represent the cutting-edge diversity in reservoir computing, each contributing to the field with distinct advantages and potential applications.

2.3.2 Hardware Implementations of Reservoir Computing

The physical implementation of RC has attracted increasing attention in diverse fields of research due to its fast speed of data processing and low learning cost [288]. Depending on different types of physical devices, various forms of RC have been widely studied, such as electronic RC [158], photonic RC [230], and atomic switch RC [266]. Among them, electronic RC have received great attention for their low-cost and flexible features. There are two main methods of realizing electronic RC: one is to implement RNNs using neural network hardware or neuromorphic computing techniques, and the other is to employ other dynamical systems instead of RNNs.

ESN and LSM are two types of RC algorithms that are based on nonlinear function neuron and spiking neuron, respectively. Both ESN and LSM models have been implemented in hardware following the first method, such as FPGA [344] and MOSFET crossbar array [158], for data recognition, classification and forecasting tasks. Both ESN and LSM models have been fully developed in FPGAs for data recognition and classification. Yi et al. [344] proposed an FPGA-based ESN model with 64 neurons. An ESN model was also implemented

in a FPGA for chaotic-time series forecasting [4]. 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% [317] on a speech recognition benchmark, the TI46 speech corpus. VLSI has also realised LSM models with 200 neurons [247] and 324 neurons [353] for speech recognition.

The second method mainly focuses on exploring different dynamical systems that are equipped with the features of reservoirs, namely high dimensionality [359], nonlinearity [359] and short-term memory [66], to serve as reservoirs directly.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 [16]. Electronic reservoirs are also being investigated for the development of low-cost machine learning devices [288]. Some examples of such dynamical systems are are built on traditional Complementary Metal Oxide Semiconductor (CMOS) devices combined with other components such as capacitor and operational amplifiers [5, 8, 16]. Memristive reservoirs outperform CMOS reservoirs in terms of circuit area and power consumption due to the nano size and energy efficiency of memristors [287, 288]. There has been research [286, 156] 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.

2.3.3 Evolution of Reservoir Computing

The evolutionary Artificial Neural Network (ANN) has attracted a large number of researchers [103, 104, 342], by which ANNs's accuracy can be further improved. Some researchers have argued that creating a reservoir at random is not sufficient for optimal per-

formance. They have suggested that a reservoir design that is tailored to a specific task will yield better results than a naive random one [180]. Hence, evolutionary reservoirs have emerged as a natural idea. Several studies have optimized reservoirs for a given application using different methods [248]. As reviewed in [17], most of the optimization and improvement efforts for reservoir computing focus on the dynamic reservoir itself. In [47], the number of nodes, reservoir connectivity, and weights have been optimized, showing that connectivity and input scaling have a significant role in the proposed approach, and that the quantity and quality of weights affect the training process.

By applying suitable evolutionary algorithms, it is possible to achieve better than average performance by selecting the right reservoir [248]. This has been done with various techniques, such as genetic algorithms [346], Evolino [246], Evolutionary Strategy (ES) [319], and Particle Swarm Optimization (PSO) [47]. Generally, there are three classes of evolutionary algorithms to optimize RC [73]. The first one is to optimize the global parameters of the reservoir, such as the spectral radius and the scaling of weights. The second one is to optimize the topology or architecture of the network directly, such as weight connections. The third one is to use a hybrid approach that combines the other two. However, there is no research on optimizing hardware or memristive reservoirs [52, 53].

By summarizing the related works of evolving reservoir computing, the conventional reservoir computing paradigm has the drawbacks that uses fixed randomly created reservoirs and conventional devices, incurring the limited accuracy and energy efficiency. Evolutionary approach and memristive architectures may offer solutions to these issues. Therefore, reservoir computing is a very promising application field for the novel EHW based on memristors, by which the optimal reservoir design could be evolved on the EHW. It further motivates the RQ3, which is *How to improve the accuracy and energy efficiency of EHW for RC?*

2.4 Benchmark Circuits

In Chapter 3, we apply voltage reference circuit, temperature sensor circuit, Gaussian function generator circuit, and pulse generation circuit, nonlinear calculation circuit as the benchmark circuits to verify our proposed approach, respectively. In Chapters 4 and 5, we apply the reservoir circuit as the benchmark circuit to verify our proposed approach.

- **Voltage reference circuit:** it is an essential component in electronic systems, providing a stable reference voltage that is unaffected by fluctuations in supply voltage or ambient temperature changes. There have also been several manual-designed voltage reference circuits, where Brokaw et al. [31] applied 29 components (BJT+resistor+capacitor) to construct the circuit, thus they still suffered the issue of large area. Evolutionary circuit design may provide possible solutions to this. In order to make our proposed approach more comparable with other approaches [152, 189, 37], the circuit setting and desired output are set to the same as theirs. We aim to output a stable $2V$ across a load resistor in conditions where input voltage varies from $4V$ to $6V$ and temperature from $0°C$ to $100°C$. The embryo circuit for evolution consists of a power supply node, an output load resistor, and ground. The performance of this circuit is quantified through a fitness function, which reflects its precision in maintaining the voltage threshold within a $0.02V$ error margin using specific transistors and resistors.

- **Temperature sensor circuit:** it is designed to transduce temperature changes into a variable voltage output. It is a critical function in various applications requiring temperature monitoring, like environmental sensing and system control. Several temperature sensor circuits have been manually designed by previous works, such as the one by Meijer et al. [191], which used 23 components (BJT+resistor+diode) to build the circuit. However, this design still faced the problem of occupying a large area. In order to make our proposed approach to be more comparable with other approaches

[152, 189, 37], the circuit setting and desired output are set to the same as theirs. Therefore, the circuit is evolved to function within a temperature range of $0°C$ to $100°C$, with the objective to maintain output voltage linearity relative to temperature changes. The preliminary circuit includes two voltage supply nodes with corresponding series resistors, a load resistor, and a ground. The evolution's success is evaluated by the squared error between the measured and expected voltage outputs.

- **Gaussian function generator circuit:** it is intended to produce an output current that forms a Gaussian distribution in relation to an input voltage. Such generators are fundamental in signal processing and communication systems where specific waveform generation is required. Previous works have also manually designed some Gaussian function generation circuits, such as the one by Popa et al. [228], which used 30 components (MOSFET) to implement the circuit. However, this design also had the drawback of taking up a large area. The target output is a peak current of $80nA$ at an input of $2.5V$. The embryo circuit used for evolution comprises a variable voltage supply with a series resistor, a fixed voltage supply, an output terminal, and ground. The quality is gauged by the congruence of the output current to the Gaussian target, with an acceptable error margin of $5nA$, and the circuit includes MOSFETs and resistors.

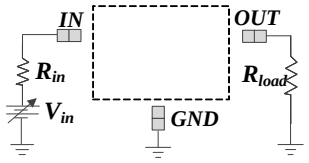
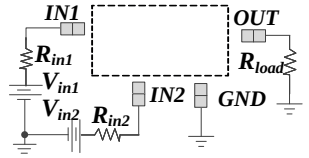
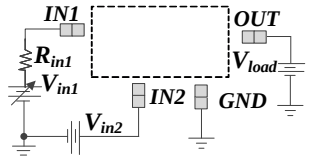
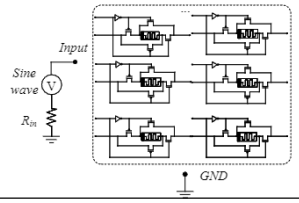
- **Pulse generation circuit:** this is a significant development in the field of neuromorphic computing [322], which seeks to emulate the neural structures and functions of the human brain in silicon to create more efficient and powerful computational systems. In neuromorphic circuits, pulses or spikes are analogous to the electrical signals used by neurons to communicate. These circuits require precise and reliable pulse generation to function correctly, mimicking the timing and variability of biological neural systems. Traditionally, pulse generation might rely on a variety of active devices like external chips [322, 265], or transistors or capacitors [27, 123], but these can result in circuits that are large, power-hungry, and difficult to integrate with dense chip designs that

modern microelectronics demand.

- **Reservoir circuit:** reservoir circuits are a key component in the field of neuromorphic computing, which aims to mimic the neural structure and processing of the human brain in hardware. These circuits are designed to replicate the dynamic, non-linear processing capabilities of biological neural networks. Yi et al. [344] proposed FPGA-based ESN, where this circuit does not use memristors and requires a significant number of transistors (37n, where n is the number of neurons). The lack of memristors implies a potential limitation in terms of adaptability and learning capabilities, as well as possibly higher power consumption due to the use of numerous transistors. Kume et al. [158] proposed Mosfet Crossbar-based ESN, where it is constructed on a PCB with 16n MOSFETs and no capacitors. While the use of a MOSFET crossbar can enhance scalability, the lack of memristors might limit the circuit's efficiency and neuromorphic capabilities. Overall, these designs reflect a balance between component choice, power efficiency, and neuromorphic capability, with memristor-based circuits showing promise for efficiency but potentially limited in functionality.

These circuits are representative of different categories of analog circuits with diverse functions and performance metrics. Each plays a crucial role in its respective domain. Therefore, the evolution of these circuits using the methods proposed in the thesis can underscore the flexibility and potential of genetic programming in automating complex analog circuit design. We also created a table to introduce the description and embryo circuits of the benchmark circuit, which is presented in Table 2.2.

Table 2.2: The descriptions of the benchmark circuits applied in the thesis.

Bechmarks	Embryo Circuit	Circuit Setting
Reference Voltage		$R_{in} = 10k\Omega$ $V_{in} = 5V$ $R_{load} = 10k\Omega$
Temperature Sensor		$R_{in1} = 1k\Omega$ $V_{in1} = 15V$ $V_{in2} = 5V$ $R_{in2} = 1k\Omega$ $R_{load} = 10k\Omega$
Gaussian Generator		$R_{in} = 1\Omega$ $V_{in1} = 2.5V$ $V_{in2} = 5V$ $V_L = 2.5V$
Reservoir Circuit		V_{in} sine wave or other signals $R_{load} = 1k\Omega$

2.5 Summary and Discussion

This chapter introduces the review of the relevant literature and background knowledge related to the thesis topic, which includes hardware platforms for Evolvable Hardware (EHW), circuit representation and evolution algorithms, and Reservoir Computing (RC) applications. It serves as the foundation and motivation for the subsequent chapters of the thesis.

In Section 2.1, the memristor, the hardware platform based on traditional devices and the hardware platform based on memristors were reviewed. The review reveals that memristors have several advantages over conventional devices, such as low power consumption, high integration density, non-volatility, and reconfigurability. These advantages make

memristors suitable for implementing novel EHW that can perform intelligent and adaptive computations. However, memristors also have some limitations, such as device variations and non-ideal effects. These limitations pose challenges for designing EHW based on memristors. Therefore, this thesis investigates novel EHW based on memristors that can overcome these challenges and leverage the benefits of memristors.

Section 2.2 provided a review of the different types of circuit representations and evolution algorithms. The main limitations of existing circuit representations include limited generality of evolutionary approach, inefficient decoding and evolutionary process. These limitations motivate the research question RQ1, which aims to develop novel circuit representations and algorithms that can overcome these challenges and enable more general and efficient approaches for circuit evolution.

However, the circuits designed by direct circuit representation are the specified circuits, which rely on a fixed circuit topology and cannot be evolved adaptively to different tasks or changed dynamically during the circuit execution. Therefore, the indirect circuit representation and its evolutionary operations will be further investigated for memristive reconfigurable architectures in this thesis. Memristive reconfigurable architectures have various types and features, requiring indirect circuit representations that can exploit their potential to evolve the memrsitive circuit based on the reconfigurable architectures. Therefore, the review of indirect circuit representation in this section identifies RQ2, which aims to investigate the evolution of a reconfigurable memristive architecture.

In Section 2.3, we provide a comprehensive review of the applications of EHW to Reservoir Computing (RC) and identify the main limitations of existing RC models and their circuit implementations. These main limitations include the long-term dependency problem and their limited energy efficiency attributed to digital or hybrid circuit implementations. This review motivates RQ3, where the novel EHW based memristor will be proposed to ad-

dress these challenges by employing the evolutionary approach in the algorithmic perspective and optimizing the memristive circuits in the hardware perspective.

In Section 2.4, we summarize the application of various benchmark circuits to validate a proposed approach in circuit design, emphasizing the use of genetic programming for automation and efficiency. The circuits discussed include a voltage reference circuit, essential for stable voltage supply in fluctuating conditions; a temperature sensor circuit for converting temperature variations into voltage changes; a Gaussian function generator circuit, critical in signal processing; a pulse generation circuit, significant for neuromorphic computing; and Reservoir Circuits, key in emulating neural structures in hardware.

Chapter Three

Analog Circuit Evolution With Improved Generality and Efficiency Using Genetic Programming[1]

Circuit representation is the first consideration in automated circuit design. It indicates how to encode a circuit. According to different data structures, there are several types of circuit representations, such as string-based [190, 178], tree-based [38, 54, 102] and graph-based circuit representations [78, 194]. However, as explained in the Section 2.2 of Chapter 2, there are also some limitations of existing circuit representations, such as complex decoding process leading to inefficiencies in the circuit evolution [190], bloat size of the evolved design [54], inefficient crossover operators and complex transformation into circuit netlists [152], and limited design ability [38].

Moreover, genetic operators such as crossover and mutation are usually representation-

[1]This chapter answers RQ1 from Section 1.1, and is based on our published papers [263, 259]. Specifically, the approach given in Sections 3.1 and 3.2.1 is based on our proposed work [263]. Section 3.3 presents the experiment part of our proposed work [263]. The approach presented in Section 3.2.2 is based on our proposed work [259]. Section 3.4 presents the experiment part of our proposed work [259]. This footnote, in addition to the statements in Section 1.2, also serves to clarify any detected overlaps between this thesis and my own published papers.

dependent and are commonly applied to random genes [152, 38]. However, randomly choosing genes during genetic operations may limit the search efficiency by loosing potentially useful sub-circuits. Moreover, due to the presence of devices with zero contribution, the bloat issue may occur, increasing the size of evolved circuit.The search efficiency could potentially be improved by using the knowledge of which part of sub-circuit is beneficial to the evolution to guide the optimisation process.

The first research question of the thesis is answered in this chapter, which aims to solve the existing problems in the circuit representation and evolutionary process:

RQ1: How to improve general applicability and efficiency of the circuit evolution?

To answer RQ1, a new circuit representation and more efficient evolutionary operations for automated analog circuit design based on GP are proposed in this chapter. Specifically, first, we introduce a new tree-based circuit representation to evolve the analog circuit, which is more efficient to be transferred into the circuit netlists and can support both two-terminal and three-terminal circuits. Second, based on this new tree-based circuit representation, we propose a novel approach that uses Shapley values to evaluate the contribution of each function node of the circuit tree to the performance of the whole tree, to guide the evolutionary process.

The rest of this chapter is structured as follows. Section 3.1 introduces our proposed circuit representation in detail. Section 3.2 explains our proposed circuit evolution algorithm using the standard Genetic Programming (GP) and the Shapley value-based GP. Section 3.3 gives the experimental studies of circuit evolution based on our proposed novel circuit representation using conventional GP. Section 3.4 gives the experimental studies of circuit evolution based on our proposed Shapley-based GP. This chapter is summarized in Section 3.5.

3.1 A Novel Tree-based Circuit Representation

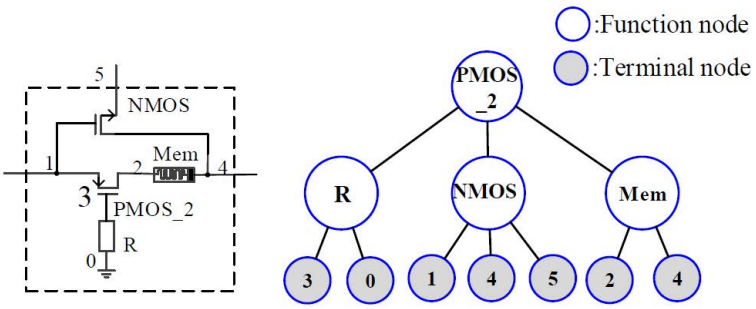
Ideally, both of the circuit topology evolution and device value optimization should be considered in analog circuit design automation, avoiding the need for expensive human design knowledge. In this section , we propose a novel tree-based circuit representation for evolving analog circuits that can be used to evolve both the circuit topology and device values. The details of representing analog circuits (Section 3.1.1), structural plausibility check to make sure each generated circuit tree is corresponding to a valid circuit (Section 3.1.2), and the advantages of the proposed tree-based representation (Section 3.1.3) will be introduced.

3.1.1 Circuit Representation and Netlist Transformation

The circuit representation is based on a multi-tree. We define the non-leaf node as function node and leaf node as terminal node (see right part of Figure 3.1).

A function node (Figure 3.2(a)) consists of two parts, the first part is a device type depending on the circuits to be implemented, for example, in a simple RC filter circuit, the device type of a function node will be R (resistor) or C (capacitor). The second part is the value tree representing the value of the circuit device in the form of a binary tree, as shown in the example provided in Figure 3.3. Every value-based device will have its corresponding value tree. For devices that do not require a value, the value tree is null.

A single terminal node (Figure 3.2(b)) refers to the position of one device port in the circuit netlist, which is represented by an integer number. According to the number of device ports, each function node has a defined arity (i.e., number of child nodes). For example, memristor, a two-port device, can be represented by an arity-2 node in the circuit representation. Similarly, MOSFET, a three-port device, can be represented by an arity-3



Device	Terminals		
	Terminal1	Terminal2	Terminal3
PMOS_2	3	1	2
R	3	0	-
NMOS	1	4	5
Mem	2	4	-

Figure 3.1: A circuit example represented by proposed method.

(a) Function node (b) Terminal node

Figure 3.2: Diagram of function node and terminal node.

node in the circuit representation. Therefore, the circuit connection of a device in the circuit netlist can be determined by assigning the position numbers of each port.

In order to enhance the flexibility of circuit representation, the devices with polarity correspond to different function nodes. For instance, the memristor and its equivalent memristor with reversed polarity in Figure 3.4(a) are represented by two different function nodes. Similarly, the PMOS with original definition and its five types of equivalent PMOS with different port definitions are also represented by fix different function nodes. In this way, the circuits with any-connection can be implemented.

During the process of converting the tree-based circuit representation into the circuit netlists, each function node needs to be specified a netlist position number. Therefore, a hierarchical relationship between parent node and child node is defined. The terminals of the parent node are formed by the left terminals of its child nodes, which allows for

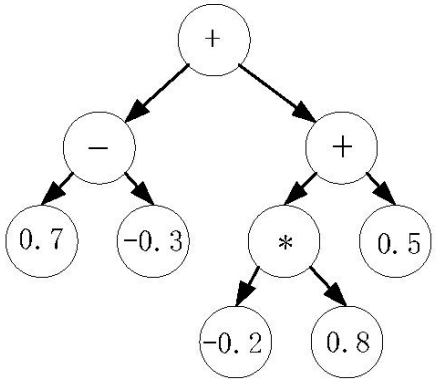


Figure 3.3: An example of value tree.

the representation to be circuit-intuitive. Figure 3.1 shows an example of how to convert the devices in a tree-based circuit representation into their corresponding circuit position numbers. In Figure 3.1, $PMOS_2$ is the parent node of R, $NMOS$, and Mem. According to the hierarchical relationship between parent node and child node, *Terminal1* of $PMOS_2$ will be *3*, which is the same as *Terminal1* of R; *Terminal2* of $PMOS_2$ will be *1*, which is the same as *Terminal1* of $NMOS$; and *Terminal3* of $PMOS_2$ will be *2*, which is the same as *Terminal1* of Mem. In this way, all the function nodes will be assigned corresponding terminal nodes, enabling the construction of a circuit netlist further.

In Koza's tree-based structure [149], both of the circuit constructing operations and the circuit devices are function nodes. However, the circuit constructing operations such as series, parallel and flip will use many nodes within a tree, which may cause some undesirable scenarios where a complex tree with plenty of circuit constructing operations and fewer circuit components can only represent a simple circuit. These undesirable scenarios can be prevented in our representation, by using different function nodes of a tree to represent circuit components and use the topology of a tree to represent the circuit connection. Therefore, we can use a simpler tree to represent the same circuit compared with Koza's tree. Figure 3.5 gives an example of encoding a given circuit by Koza's and our method. We can see that many circuit construction operation nodes are used by Koza's method to construct the circuit, such

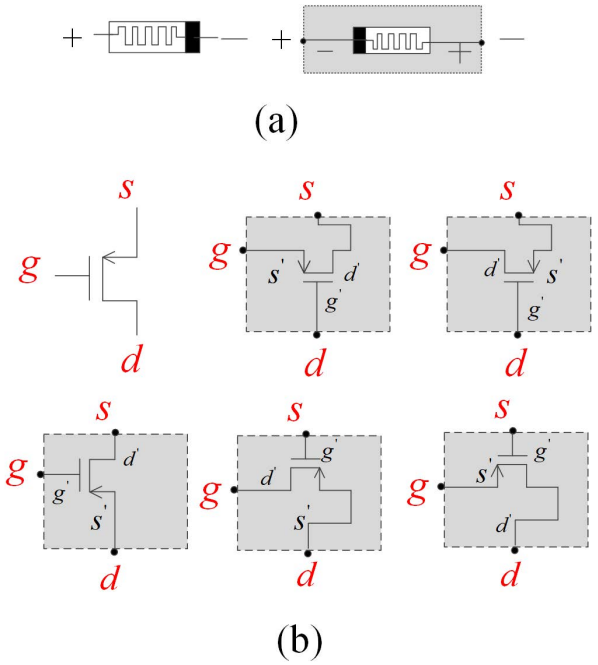


Figure 3.4: Diagram of equivalent devices with different terminal definitions. (a) Memristor and its equivalent device with reversed polarity (b) PMOS and its five types of equivalent device.

as S (series), F (flip) and E (end). As for our proposed method, only the circuit devices and their netlist positions are necessary to construct the circuit, which is therefore more compact than Koza's tree.

Besides the above tree structure representing the circuit topology, the value tree, which is embedded in the related function nodes, is designed to represent the device value. In the value tree, the arithmetic operators are regarded as function nodes, and the terminal nodes are float numbers from -1 to 1. The equation to calculate the value of a tree is:

$$Value_{device} = A \times (10^{Value_{tree}}), \tag{3.1}$$

where A is the fitting parameter that is capable of scaling the obtained value into the reasonable range of corresponding devices, and $Value_{tree}$ can be calculated by the arithmetic operators and float numbers in the value tree. Taking Figure 3.3 as an example and assuming

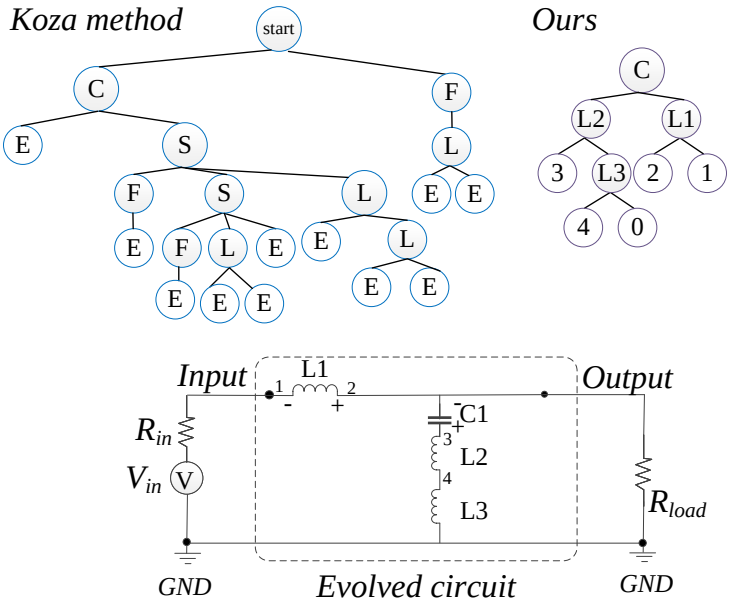


Figure 3.5: An example of encoding the same circuit by Koza's and our methods.

$A = 10^5$, the $Value_{tree}$ of a resistor is calculated as:

$$Value_{device} = 10^5 \times (10^{((-0.2*0.8)+0.5)+(0.7-(-0.3))}). \tag{3.2}$$

The value tree allows the approach to optimise by shuffling around arithmetic operations from a small and tractable set, as opposed to randomly having to "mutate" values directly. This can protect those "good" arithmetic operations that are beneficial for the evolution, generating better value gradually during the evolution.

3.1.2 Three Types of Structure Check

We want to prevent the bloat problem [267] of tree structure and the invalid circuits that will cause simulation failure, therefore, three types of structure check are applied in this work. In order to check if a circuit is feasible, the usage count of different terminal nodes is an important indicator. Algorithm 3.1 gives how to count the terminal nodes of a tree,

where the post-fix traversing is applied to each parent node. When the current traversed parent node is the terminal node, the current parent node will be added to the terminal set (Ln 2-3 in Algorithm 3.1). When the function node is the current parent node traversed, all its child nodes will be traversed. The value on the terminal node will be stored in *portList*. The *terminal_count* will be updated with the counts of each element in *portList* (Ln 8-9 in Algorithm 3.1). Three types of structure check are introduced accordingly.

First, our representation considers that there is a predefined embryo circuit representing the initial circuit configuration of the to-be evolved circuit. The embryo circuit design is generally simple, which could be specified depending on different circuit requirements [152, 37, 190]. The common embryo circuit includes voltage sources, ground, and load resistor. This part of circuit will not be evolved, i.e., will remain fixed during the evolutionary process. However, the way with which the to-be evolved circuit will be connected with the embryo will be determined by the evolutionary process. The external nodes of embryo circuit are necessary nodes. We also define the unnecessary nodes from two aspects, which are node counts and node function. Specifically, a node that has been assigned more than twice or does not belong to the embryo circuit is an unnecessary node. The first step is that the external terminals of the embryo circuit (necessary nodes) should be checked if they are all assigned to the tree terminal node by the evolutionary process. If they are not, the unconnected terminal of embryo circuit will be isolated, incurring invalid circuits during the evolution. Such infeasible individuals are repaired by replacing a random unnecessary node by the isolated necessary node. The *step 1* in the Figure 3.6 gives an example of the first structure check. Nodes $0, 1, 2, 3$ indicates the embryo nodes that must be assigned in the tree necessarily. However, as we can see that the node 1 has not existed in the tree. Therefore, the unnecessary nodes, which are $5, 7, 9, 6, 8, 15, 12$, one of them will be selected randomly and replace it by node 1. In this way, all the external terminals of the embryo circuit will be connected to the evolved circuit.

Algorithm 3.1 Pseudo code of nodes count function *update_terminal*()

Input: *parentnode* that will be taken the *update_terminal*.
Output: *parentnode* after *update_terminal*.

```
1: def update_terminal(parent : Node):
2: if parent.type == 'terminal' then
3:    terminal_set.append(parent)
4: else
5:    if parent.type == 'function' then
6:       for node in parent.childList do
7:          update_terminal(node, Calparent)
8:          for port in node.portList  do
9:             terminal_count[port]+ = 1
10:         end for
11:      end for
12:   end if
13: end if
14: Parent node with updated terminals
```

Second, in our circuit representation, the terminal nodes represent the port position. Therefore, we need to make sure the devices are connected to each other composing a complete circuitry and preventing the problem of hang terminals, i.e., isolated terminals of devices that are not connected to the evolved circuit. To prevent this problem, one terminal node should be assigned at least twice in a single tree. Therefore, we need to ensure that the terminal nodes randomly generated in the tree-based circuit representation appear in the tree no less than twice. The *step 2* in the Figure 3.6 shows an example of the second structure check works. Nodes $4, 11, 10, 13, 14$ are dangling terminals without connections with other nodes. The last node 13 will be fixed firstly, where a random node from unnecessary nodes and embryo nodes will substitute the node 13. Then, the neighboring two nodes will be set as the same. As shown in Figure 3.6, $4, 11, 10, 13, 14$ are changed to $4, 4, 10, 10, 8$, respectively. By the tree structure check, all the embryo nodes could be connected into the evolved circuit and there will be no dangling terminals existing in the evolved circuit.

Some researchers claimed that the bloat problem of tree can be improved by limiting the tree depth or the number of nodes [236]. The third type of structure check is to apply a

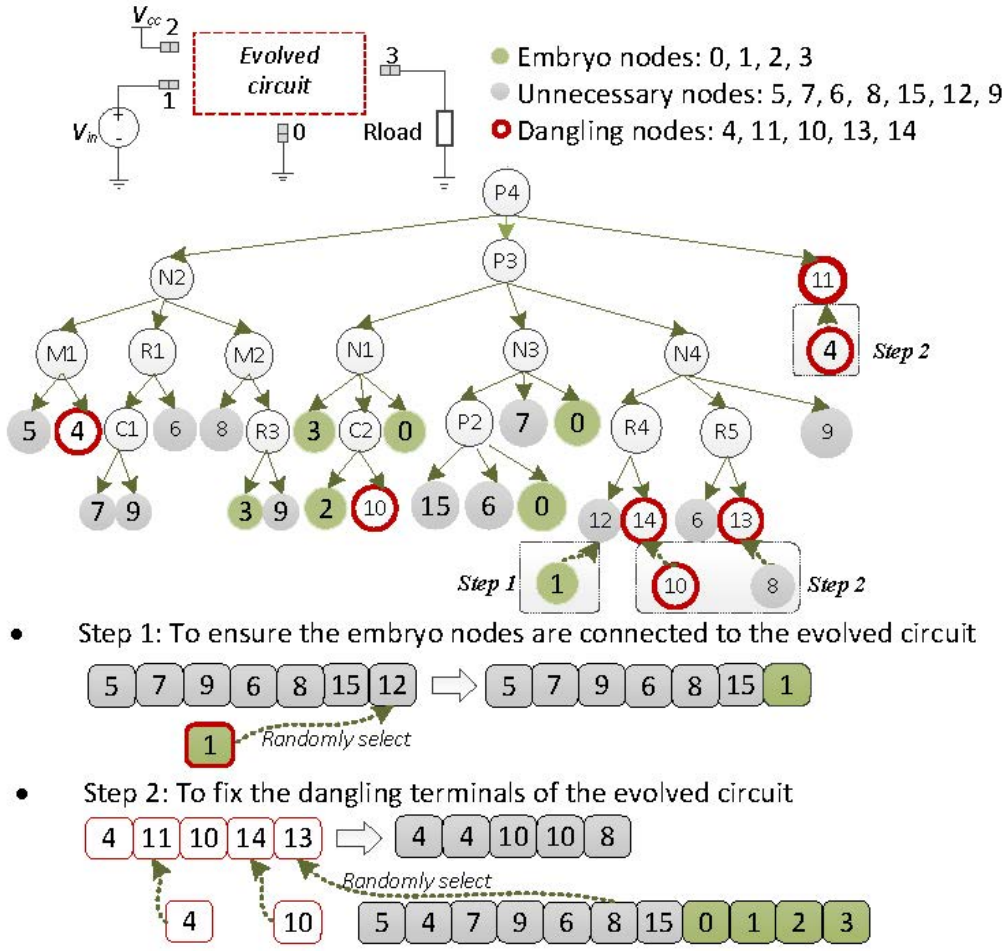


Figure 3.6: Diagram of tree structure check.

max depth limit for the tree. Therefore, the tree depth will be restricted in the range between minimum and maximum. After generating a new tree, the tree depth will be checked and the part that exceeds the max depth will be replaced by its left terminal node.

3.1.3 Advantages of Our Proposed Tree-based Representation

The advantages of our proposed tree-based representation are as follows:

• Efficient transformation into circuit netlists

In Koza's tree representation [150], function nodes contain both of the circuit construction

operations, such as parallel (P) or series (S) operations, and device types, such as resistor (R) or capacitor (C), which leads to complex tree structures. Compared with Koza's tree representation, the function nodes of our proposed tree-based representation only contain circuit devices manipulating different ways of connections between function nodes and terminal nodes (port position) to represent circuits, which leads to more compact tree structure. We only consider the two-terminal and three-terminal devices are applied in this work. The same height H for both of our proposed tree-based representation and Koza's tree are given for the analysis. When the circuit devices are all the two-terminal devices, there will be $\frac{2^H-2}{2}$ devices, and when the circuit devices are all the three-terminal devices, there will be $\frac{3^H-3}{6}$ devices. Therefore, the number of circuit elements that our proposed tree can represent will be in the range of $\left(\frac{2^H-2}{2}, \frac{3^H-3}{6}\right)$. However, as the function nodes of Koza's tree not only contain the device types but also the circuit construction operations, the number of the circuit elements that Koza's tree can represent will be in the range of $\left(\lambda \times \frac{2^H-2}{2}, \lambda \times \frac{3^H-3}{6}\right)$, where λ is a constant indicating the proportion of the number of device types to all the number of function nodes. Theoretically, λ could be 1, indicating the function nodes are all the circuit elements. Actually when $\lambda = 1$, there will be only one device in the circuit generated by Koza's method, which is unlikely to perform the desired function of the circuit. Therefore, $\lambda \in (0, 1)$. As a results, our proposed tree-based representation is more compact, which means it can represent more devices in a circuit with the same tree height. More compact tree structures make transformation into netlists more efficient.

In addition, the form of "Device–circuit position" of our proposed representation also leads to more efficient transformation into netlists. Specifically, the executable circuit netlist could be transformed from the form of "Device–circuit position" of our proposed representation by the post-order traversing. However, Koza's tree-based representation [152] has tree structures based on the form of "Circuit construction operations–devices", which require extra steps to be executed in addition to traversing the whole tree in order to assign the

corresponding terminal position to circuit devices for constructing or updating the circuit netlist. Therefore, considering the more compact tree structure and the form of "Device–circuit position", our proposed tree-based representation can make the transformation into circuit netlist more efficient.

•Support for both of two-terminal and three-terminal devices

As mentioned in Section 3.1.1, the proposed representation can be used both with two-port (resistor or capacitor) and three-port (transistor) electrical devices. Different from topology-restricted methods [38], the proposed circuit representation can be applied in both of two-port and three-port based topologies. Therefore, the proposed method has a wider application range of circuit design.

• Suitable and efficient crossover operators for circuit evolution

Some researchers proposed that it is challenging to design crossover operators for graph-based circuit representation due to its close-loop structure [197]. Thus, only the point mutation operations are applied in some work [78]. Compared with graph-based representations, the proposed circuit representation lends itself better to crossover operators, which may help the evolutionary process to find better circuits. The sub-circuits of a circuit can be represented by the branches of the tree in our proposed representation, facilitating the design of suitable crossover operators for circuit evolution. However, in Koza's tree representation [152], the sub-circuits of the whole circuit are represented not only by their corresponding branches but also by the function nodes in the parent level of the branches, which may lead to inefficient crossover operations. The crossover operator adopted with our representation is explained in Section 3.2.1.

3.2 Evolving Analog Circuit Using Genetic Programming

In order to improve the design ability and the efficiency of circuit netlist transformation in the circuit evolution, we proposed a novel tree-based circuit netlist in last section. In this section, how the analog circuits are evolved using GP based on the proposed novel tree-based circuit representation will introduced. Based on our proposed novel tree-based circuit representation, the conventional GP and the Shapley value-based GP have been designed to the analog circuit evolution in Section 3.2.1 and Section 3.2.2, respectively.

3.2.1 Conventional Genetic Programming for Circuit Evolution

Overall Flowchart of conventional GP

As shown in Figure 3.7, a GP-based algorithm for evolving analog circuits contains the following steps:

Step 1: Parameter settings – a series of parameters is set, such as population size N, tournament size T, max iterations M, max depth of tree H, topology crossover rate P_{cross}, mutation rate P_{mutate} and value crossover rate $P_{valuecross}$.

Step 2: Population initialization – According to the parameters set in Step 1, the population is initialized. The details of population initialization in our proposed GP design are introduced in Section 3.2.1.

Step 3: Fitness evaluation – Each individual is assigned a fitness value by the proposed fitness function. The specified fitness functions for different tasks are proposed in Section 3.3.

Step 4: Elitism strategy – The fourth step is to employ the elitism strategy. All the individuals in the population are sorted by their fitness value and the one with the best fitness is reserved.

Step 5: Topology crossover – The next step is topology crossover. We select two parents by n-tournament selection to perform topology crossover with probability P_{cross}. One child is generated as a result of the crossover operation, and will survive to the next generation. Moreover, the generated child will replace the parent 2 to execute value crossover operation with parent1. The detail of the topology crossover is explained in Section 3.2.1.

Step 6: Value crossover – The next step is value crossover. Two parents are selected by n-tournament selection. Two function nodes will be selected from the parents respectively and go through value crossover with probability $P_{valuecross}$. The resulting child node of the value crossover will be taken as the corresponding function node of Parent 1, replacing Parent 1's previous function node. This crossover operator is introduced specifically in the following part of Section 3.2.1.

Step 7: Mutation – The last step is mutation. One of two mutation operations (delete or add) is randomly selected with equal probability to being executed on the *i-th* individual. The mutation operators are introduced in the following part of Section 3.2.1.

Circuit Topology Evolution

Population initialization defines how the individuals in the initial population are generated and how to ensure that their structure is valid. Each tree individual is generated by the *grow* method [217], which means that the distances between each leaf nodes and the root node are not the same.

The three types of structure check mentioned in Section 3.1.2 are applied to prevent invalid circuits.

Based on the proposed circuit representation, circuit topology evolution is implemented by executing the operations of topology crossover and mutation. An example of topology crossover operation based on the proposed circuit representation is shown in Fig-

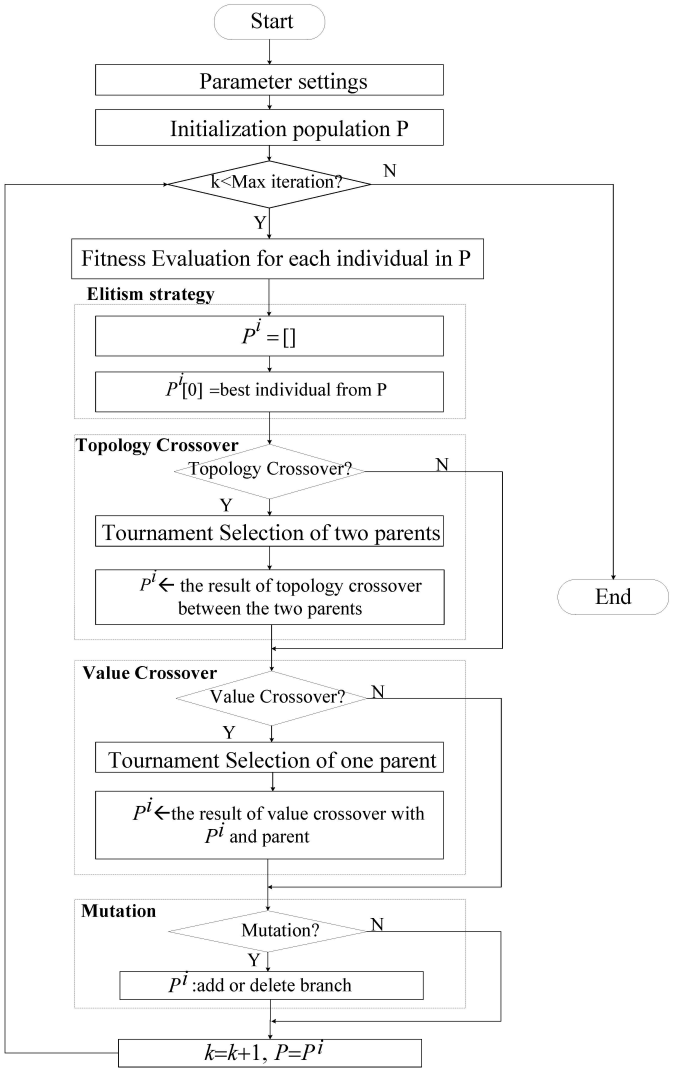


Figure 3.7: Flowchart of GP algorithm for evolving analog circuits.

ure 3.8. The crossover operation is performed among non-leaf nodes. As shown in Figure 3.8, we select two parents from T individuals by n-tournament strategy. A node from the first parent and a node from the second parent are randomly selected, e.g., terminal node 2 and node C_1 in Figure 3.8 and node C_1 of Parent 2 is also randomly selected. Then the sub-tree from the second parent rooted at the selected node is used to replace the sub-tree from the first parent rooted at the selected node to generate a child, which will be added into the population.

Figure 3.9 shows an example of mutation operation that explains how the mutation operation is executed based on proposed circuit representation. Two types of mutation operations are applied in this work, which are *delete* and *add*, respectively. As for the *delete* operator, a subtree is randomly selected to be deleted and the left-most terminal node of this sub-tree is used to replace it. As for the *add* operator, a terminal node from the current tree is selected uniformly at random, and then a randomly generated sub-tree with a feasible depth is created to replace the selected terminal node. "Feasible depth" here means that its depth must be between the minimum and maximum depth parameter of the algorithm. This is implemented by the three types of structure checks introduced in Section 3.1.2. The upper part of Figure 3.9 shows an example of *delete* operation, in which the function node R_1 is randomly selected to be deleted and its left terminal node *3* will substitute the node R_1 itself. The bottom part of Figure 3.9 shows an example of *add* operation, in which a random terminal node *1* is substituted by a newly generated sub-tree.

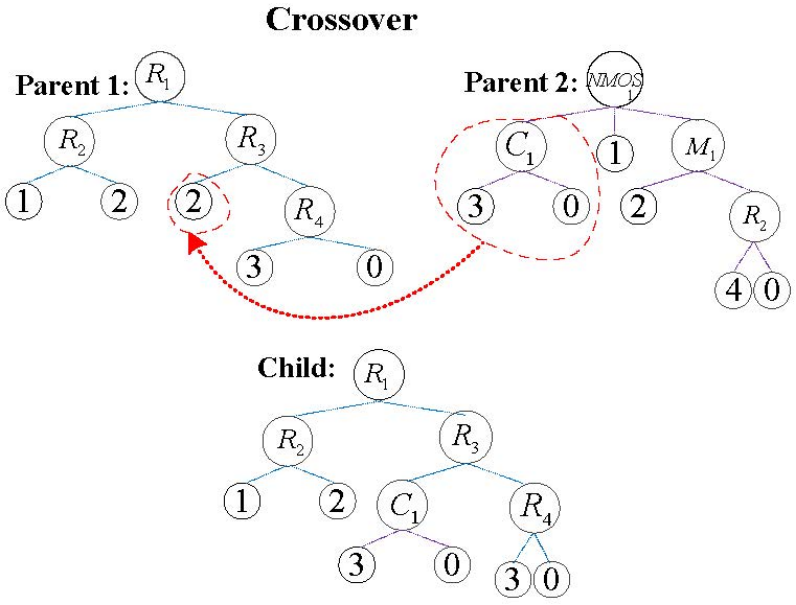


Figure 3.8: An example of topology crossover operation of our proposed GP algorithm.

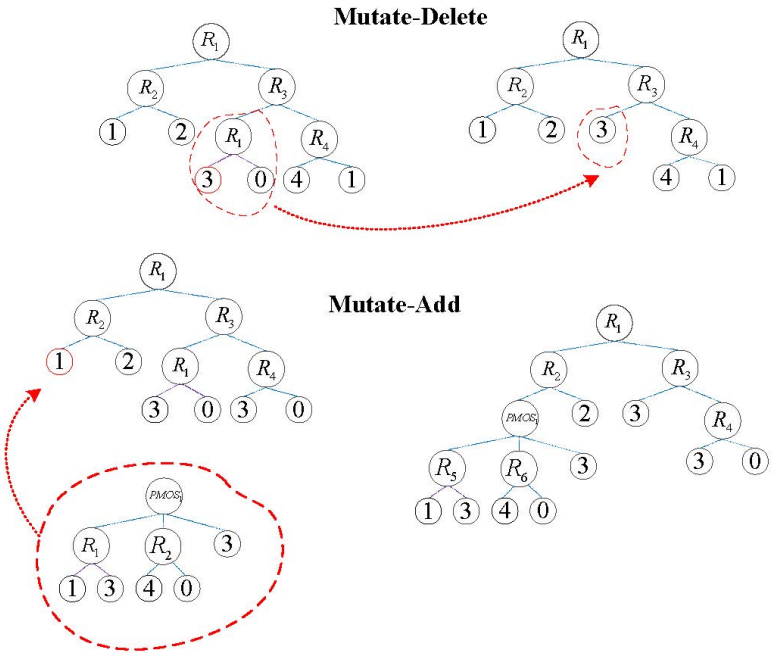


Figure 3.9: An example of mutation operation of our proposed GP algorithm.

Device Value Optimization

Besides the circuit topology, the device values of the circuits also need to be determined automatically for the circuit design. In this work, the device value is represented by the value trees (proposed in Section 3.1.1), which are embedded in the corresponding function nodes. The device value is evolved by the operation of value crossover alongside the evolution of the circuit topology as outlined in the flowchart from Figure 3.7. The value crossover operation can only be executed to two function nodes of the same type of device. Here, the "same type" refers to the same device type of function nodes. For example, the value of a resistor could only go through value crossover with the value of another node that also represents a resistor. Similarly, a node representing a capacitor cannot go through value crossover with another node representing a resistor.

Figure 3.10 shows the detail that how the value crossover works. R_1 is a randomly selected function node in the *i-th* individual. R_2 is a randomly selected function node of

Parent 1 (which was selected by tournament selection) among the function nodes of the same type as R_1. If no function node of the same type exists, then another function node could be selected until there is the one with the same type. Assume that both nodes correspond to a resistor. Their corresponding value trees go through crossover as follows. A sub-tree is randomly selected for each of the two value trees. The sub-tree from the value tree of R_2 then replaces the sub-tree from the value tree of R_1. In this example, after the value crossover is applied, the value tree of R_1 changes from $169.8K\Omega$ to $70.8K\Omega$ according to Equation 3.1.

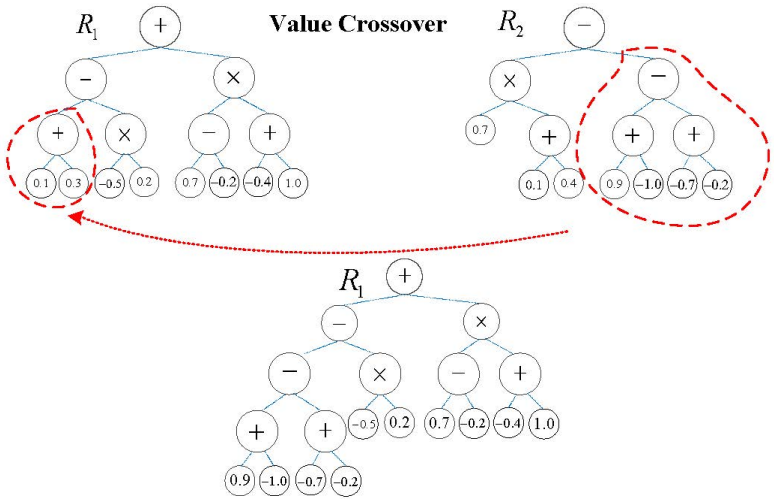


Figure 3.10: An example of value crossover for a resistor.

As mentioned above, evolutionary design of analog circuit contains both of circuit topology evolution and device value optimization, which allows for the diversity of evolved circuits and better evolution results. The specific genetic operators (crossover and mutation) are proposed for the circuit topology evolution and device value optimization. The sub-circuits of a whole circuit can be represented by the branches of a tree. Therefore, the crossover operation executed on the tree stands for the exchanging between sub-circuits. This is an advantage over graph-based representations, for which suitable crossover operators are difficult to design. Mutation operations include delete and add, which can increase and reduce the depth of a tree, allowing for increasing the diversity of circuit topology.

3.2.2 Shapley Value-based Genetic Programming for Circuit Evolution

Preliminaries on Shapley Values

In Cooperative Game Theory (CGT) [30], N players are related to each other through a score function $V : 2^n \to \mathbb{R}$, where $V(S)$ is denoted to the performance of the model after all elements in $N \setminus S$ are zeroed out. For that, Shapley value [237] is introduced as an equitable way of sharing the group reward among the players. It is based on trying to answer the question: how much does player i contribute to the coalition? The marginal contribution $\Delta_{V(i,S)}$ of player i with respect to a coalition S is defined as the additional value generated by including i in S:

$$\Delta_V(i, S) = V(S \cup i) - V(S) \tag{3.3}$$

Intuitively, the Shapley value can be understood as the weighted marginal contributions to every potential subsets of players. In terms of the subsets S and the number of permutations for which some ordering of S immediately precedes player i, the Shapley value could be written as:

$$u_i = \frac{1}{|N|} \sum_{S \subseteq N - \{i\}} \Delta_v(i, S) / \begin{pmatrix} |N| - 1 \\ |S| \end{pmatrix} \tag{3.4}$$

This formula takes into account the interactions between players. As a simple example, suppose there are two players that improve performance only if they are both present or absent and harm performance if only one is present. The equation considers all these possible settings.

Some researchers have developed Shapley value to the feature attributions, of which results get higher consistency with human intuition [181]. Second, some researchers have adopted Shapley value to measure sample importance, computing the contribution of each

training example [132] [82]. Third, Shapley values have also been used to compute the contribution of individual elements of a machine learning model e.g. neurons in a neural network model. Researchers have computed the Shapley value of neuron for exploring the neural network topologies and pruning small models [278].

Shapley Value-based Tree and Subtree Evaluation

Consider the circuit tree representation explained in Section 3.1. For a tree T containing a function node set N_F, in order to identify the contribution of each function node $node_i \in N_F$ to the overall performance $V(T)$ in a circuit-plausible way, we need a suitable measurement method. This measurement denoted as u should satisfy the following properties, where $SubTree(\cdot)$ is a function converting a node to its sub-tree:

1. **Zero contribution:** In the circuit evolution, one decision to make is how to handle circuit devices/sub-circuits that have no contribution. Therefore, the measure u should be capable to identify these devices/subcircuits. If $node_i \in N_F$ has no effect on performance when combined with any other $node_j$, it should be assigned a zero value. More precisely, for $\forall node_j \in N - node_i$, if $V(SubTree(node_i) \cup SubTree(node_j)) = V(SubTree(node_j))$, $u_i = 0$;

2. **Symmetry:** In the circuit evolution, there are some circuit devices/subcircuits that generate the same change in the performance. These could be referred to as symmetric elements. The measure should be capable of identifying these devices/subcircuits. If $node_i$ and $node_j$ always generate the same change in the performance when combined with any other $node_k$, then $node_i$ and $node_j$ should be assigned the same value by symmetry. More precisely, for $node_i$, $node_j$ and $\forall node_k \in N_F - \{node_i, node_j\}$, if $V(SubTree(node_k) \cup SubTree(node_i)) = V(SubTree(node_k) \cup SubTree(node_j))$, $u_i = u_j$;

3. **Additivity:** To evaluate the contribution of a subcircuit, we need to be able to define its contribution as the sum of the separate contributions of each of its devices. More precisely, for a function node subset S, if the overall fitness $V(SubTree(S))$ is the sum of each separate performance $V(node_i), node_i \in S$, the value u_S should be the sum of its value for each $node_i \in S$:

$$u_S = \sum_{i=0}^{|S|} u_{node_i}. \tag{3.5}$$

$$u_{node_i} = \frac{1}{|N_F|} \sum_{S \subseteq N_F - \{i\}} \left(V(SubTree_{S \cup node_i}) - V(SubTree_S) \right) / \left(|N_F| - 1 \, |S| \right) \tag{3.6}$$

We write $V(S)$ to denote the performance of subtree S. In our context, u_{node_i} is the Shapley value representing the contribution of the subtree rooted by $node_i$. The Shapley formula in Equation 3.6 uniquely provides an equitable assignment of values to nodes. Computing Shapley, however, requires computing all the possible marginal contributions which is exponentially large in the node number. Here, we introduce a truncated Monte Carlo Shapley [83] for circumventing this problem. First, we sample a random permutation of function nodes and generate the sub-tree set π. Then, we scan the permutation from the first to the last element and calculate the marginal contribution of each element. The scan process is truncated when the fitness V_j of subtree j is within a pre-defined performance tolerance of the overall fitness $V(T)$ and set the marginal contribution to zero for the rest of the elements in this permutation. By repeating this same procedure multiple times until the convergence criterion is met, the final result gives an unbiased estimate of the Shapley value. The pseudo code for the truncated Monte Carlo Shapley value procedure for a circuit tree is given in Algorithm 3.2. The convergence criteria is set as the maximum number of iterations.

- **The subtree π_j has the overlapped nodes with the subtrees $\pi_i \in \pi, i < j$:** For

Algorithm 3.2 Pseudo code of Truncated Monte Carlo Shapley of a circuit tree *shapley_value*()

Input: T subtrees that will ben taken the *shapley_value*().
Output: The Shapley values of the current subtree T.

```
def shapley_value(T : tree):
N_F ← function node set of T
Initialize v_i = 0 for i = 1, ..., |N_F|
while Convergence criteria not met do do
    t ← t + 1
    π : Random permutation of sub-trees with function node ∈ N_F as root
    V_0^t ← V(∅)
    for j ∈ {1, ..., |N_F|} do
        if |V(T) − V_{j−1}^t| < Performance Tolerance then
            V_j^t = V_{j−1}^t
        else
            V_j^t = V({π_1^t ∪ ... ∪ π_j^t})
        end if
        u_{π_j^t} ← ((t−1)/t) u_{π_j^{t−1}} + (1/t)(V_j^t − V_{j−1}^t)
    end for
end while
```

$\pi = \{\pi_j\}$ $(j = 1, ..., n)$, the performance of $\{\pi_1 \cup ... \cup \pi_j\}$ can be calculated by:

$$\begin{aligned} V(\{\pi_1 \cup ... \cup \pi_j\}) = f(Vol_{H_q(\{\pi_1 \cup ... \cup \pi_j\})}, Vol_{Target}) \\ -f(Vol_{H_1(\{\pi_1 \cup ... \cup \pi_j\})}, Vol_{Target}), j = 1, ..., n. \end{aligned} \tag{3.7}$$

Where $f(\cdot)$ is the fitness function, Vol indicates the voltage of one circuit node. Here we use voltage as the circuit measurement, and the current flowing across the circuit node also can be used as the circuit measurement. And Vol_{Target} denotes the target voltage of the evolutionary circuit design. Based on the circuit tree representation in Section 3.1.1, $H_q(\{\pi_1 \cup ... \cup \pi_j\})$ represents the last terminal node of a function node, where q is the last terminal of a function node. We usually apply this node as the output terminal of subtree π_j. $H_1(\{\pi_1 \cup ... \cup \pi_j\})$ represents the first terminal node of a function node. We usually apply this node as the input terminal of subtree π_j. Therefore, the first term on the right side of the equal sign indicates the differences

between the voltage on the input terminal of π and the target voltage, and the second term on the right side of the equal sign indicates the differences between the voltage on the output terminal of π and the target voltage. The differences of these two terms represent how much this sub-circuit decoded by π contributes towards achieving the target voltage.

- **The subtree π_j has no overlapped nodes with the subtrees $\pi_i \in \pi, i < j$:** For $\pi = \{\pi_j\}$ $(j = 1, ..., n)$, if the circuit counterpart of π_j is not connected with that of $\{\pi_1 \cup ... \cup \pi_{j-1}\}$, its performance can be calculated by:

$$V(\{\pi_1 \cup ... \cup \pi_j\}) = max(V(\{\pi_1 \cup ... \cup \pi_{j-1}\}), V(\pi_j)), \quad j = 2, ..., n. \tag{3.8}$$

Combined with Algorithm 3.2, under the calculation scheme of the max function in Equation 3.8, the subtree π_j which cannot bring performance improvement compared with the subtrees $\{\pi_i, i < j\}$, will get $O_j - O_{j-1} = 0$. So that the increment of u_j will be zero, which means the contribution of π_j is also zero.

Circuit Evolution Oriented by Shapley Value

•Overall Flowchart of the Shapely Value-based GP

The overall flowchart of evolving analog circuits is shown in Figure 3.11. It contains 9 major steps: parameter settings, population initialization, transformation to the SPICE netlist, circuit simulation, fitness evaluation, elitism strategy, topology crossover, value crossover, and mutation.

The first step (parameter settings) sets the population size N, tournament size T, max iterations $Max\_Iteration$, shapley iterations $Shapley\_Iteration$, max depth of tree

D, topology crossover rate P_{cross}, mutation rate P_{mutate} and value crossover rate $P_{valuecross}$. Then, the population are initialized based on the our proposed circuit representation from Section 3.1.1. Then, the initialized tree individuals are transformed into the circuit netlist, based on which the circuit is simulated. Based on the results of the circuit simulation, we evaluate the fitness of each individual in the population. The individual with the best fitness is saved in $P^{new}[0]$ according to the elitism strategy.

After that, the evolution has two possible stages. If the number of iterations is within a threshold $Shapley\_iter$, the first evolutionary stage is triggered (orange box). To stimulate diversity, this stage applies genetic operators (topology crossover, value crossover, and mutation) to random nodes as in previous work [263]. If the number of iterations is above $Shapley\_iter$, the second evolutionary stage is triggered (green box). This stage adopts Shapley-guided genetic operators to guide the evolutionary process towards more promising regions of the search space.

In both the first and second stages, we select two parents by n-tournament selection to perform topology crossover with probability P_{cross}. And by the probability, $P_{valuecross}$, the individuals will taken the value crossover operation, where it can only be executed to two function nodes of the same type of device.

•Shapley-oriented Mutation Operations

Figure 3.13 gives an example of Shapley-oriented mutation operators, where the function node with the worst Shapley value R_4 will undergo either the *delete* or *add* mutation operation. The subtrees with lower Shapley value will be removed, as for the *delete* operator, it may alleviate the increasing size of the tree with less performance degradation. And as for the *add* operator, it also provides the possibility that the added part will beneficial to the evolution.

•Shapley-oriented Crossover Operations

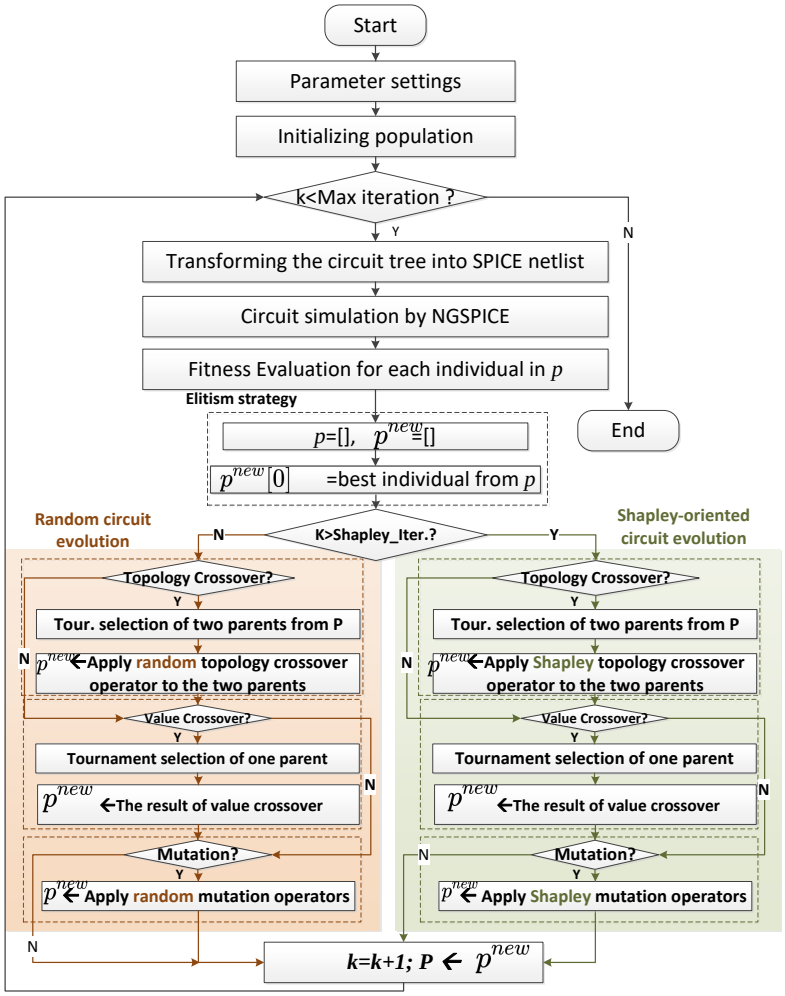


Figure 3.11: Overall flowchart of evolving analog circuits.

Figure 3.12 gives an example of Shapley-oriented crossover operator, where the crossover will be performed between the subtrees rooted by the function node with the best Shapley value C_1 in Parent 2 and the function node with the worst Shapley value R_4 in Parent 1, and their resulting child tree will be added to the next generation.The subtrees with higher Shapley value will not be destroyed by crossover and can be inherited to the next generation, making the evolution toward more promising space.

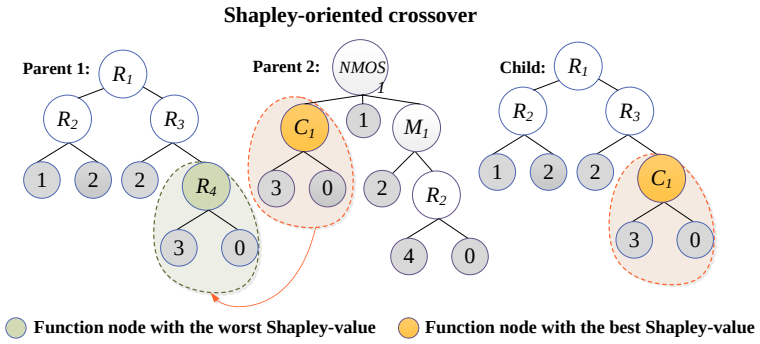


Figure 3.12: An example of Shapley-oriented topology crossover operation.

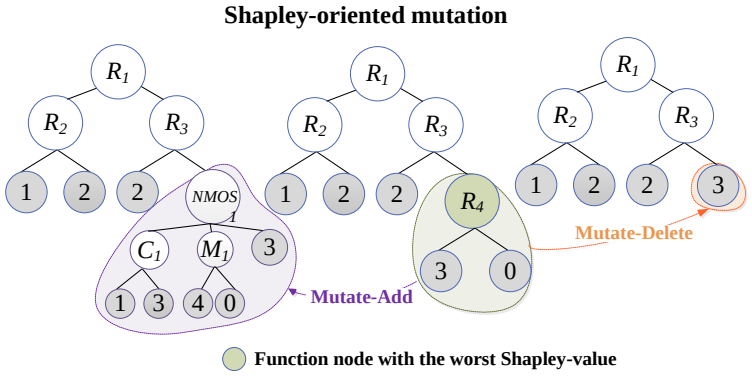


Figure 3.13: An example of Shapley-oriented mutation operation.

3.3 Experimental Studies for Novel Tree-based Circuit Representation Using Conventional GP

This section presents the experimental studies of our proposed novel tree-based circuit representation using conventional GP. Specifically, Section 3.3.1 introduces the experimental setup, Section 3.3.2 presents the validation results of crossover operators and feasibility checks, Section 3.3.3 gives the comparison results under different FEs, Section 3.3.4 presents the comparison results with other existing approaches, Section 3.3.5 presents evolution results of a memristor-based pulse generation circuit.

3.3.1 Experimental Setup

We implemented our GP in Python. We also use Python to generate netlists. The implementations will be made available as open source in GitHub [2]. The performance of the evolved circuits is evaluated in simulation using NGSPICE [310], which is based on Spice3 [211].

Three benchmark circuits are chosen to evaluate the proposed approach, namely voltage reference circuit, temperature sensor circuit, and Gaussian function generator, respectively. These benchmark circuits are widely applied to evaluate the automated circuit design methods [150] [190] [37]. In this section, we show how our proposed method can be applied to evolve these three benchmark circuits and compare the results with existing approaches. The circuit evaluation of all the compared work are carried on the NGSPICE. In order to make a fair comparison, all the fitness values are computed by the same fitness function proposed in [150], and will be explained in Section 3.3.1. Moreover, the related parameters are listed in Table 3.1.

Experimental Setup for Voltage Reference Circuit

Our first experiment is to evolve a voltage reference circuit, which is to generate a fixed output voltage $V_{out} = 2V$ on the load resistor when the input voltage varies within the interval $4V \leq V_i \leq 6V$ and the circuit temperature varies within the interval $0°C \leq T \leq 100°C$. All the candidate circuits are simulated with DC sweep, where the intervals of the input voltage DC sweep are $0.1V$ and the intervals of the temperature sweep are $25°C$. Therefore, there will be 21 discrete values of input voltage and 5 discrete values of temperature, giving a total of 105 measured points for the DC sweep simulation.

- Embryo Circuit: The embryo circuit of voltage reference circuit is shown in Figure 3.14.

[2] https://github.com/embeddedsky/EvoCkt.git

There are three accessible nodes, which is one power supply (with a $1K\Omega$ input resistor), one output load resistor ($10K\Omega$), and ground. The part marked by the dashed line will be further evolved.

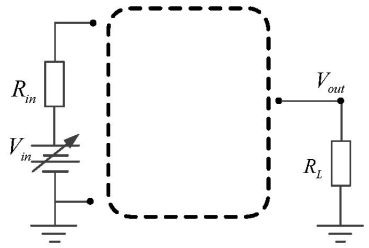


Figure 3.14: The diagram of embryo circuit for voltage reference circuit. Where $R_{in} = 1K\Omega$, load resistor $R_L = 10K\Omega$, and the output voltage V_{out} will be measured to be evaluated.

- Fitness Function: Under the different circuit temperatures T_i, each measured output voltage point $V_{outi,j}$ has its corresponding target value $V^*_{outi,j}$, where i denotes the i-th circuit temperature, and j denotes the sampled points of output voltage. The fitness function is defined as the following equation [150]:

$$fitness = -\sum_{i,j} \varepsilon_{ij}, \tag{3.9}$$

where ε_{ij} is :

$$\varepsilon_{ij} = \begin{cases} \left(V_{outi,j} - V^*_{outi,j}\right)^2, \text{if } |V_{outi,j} - V^*_{outi,j}| \geq 0.01V \\ \\ 0, \qquad\qquad \text{if } |V_{outi,j} - V^*_{outi,j}| < 0.01V. \end{cases} \tag{3.10}$$

- Device Set: The devices used for evolving the voltage reference circuit are the same as those used by the baseline approaches, namely NPN (BC846B), PNP (BC856B) bipolar junction transistor, and resistors.

- Circuit Quality Measurement: "Hit" is applied in this work to measure the evolved circuit quality, which has been widely used in previous work [150] [190] [37]. "Hit"

refers to the situation where the absolute difference between the measured point of output voltage and its target value (*error*) is less than or equal to $0.02V$. Therefore, the proportion of the number of "Hit" to the total number of measured points indicates the quality of the evolved circuit.

Experimental Setup for Temperature Sensor Circuit

Our second evaluation task is to evolve a temperature sensor circuit, of which output voltage changes with the different circuit temperatures. The variation range of temperature is $0°C \leq T \leq 100°C$, and the variation range of output voltage is $0V \leq V_i \leq 6V$. All of the candidate circuits will be simulated with temperature sweep. The sample step of temperature is $5°C$, giving 21 measured points for the sweep simulation.

- Embryo Circuit: The embryo circuit for evolving temperature sensor circuit is shown in Figure 3.15. There are four accessible nodes, where one is the positive voltage supply (V_{in1} with a series resistor R_1), one is the negative voltage supply (V_{in2} with a series resistor R_2), one is the output terminal (load resistor R_L), and final one is ground. The part marked by the dashed line is to-be evolved circuit.

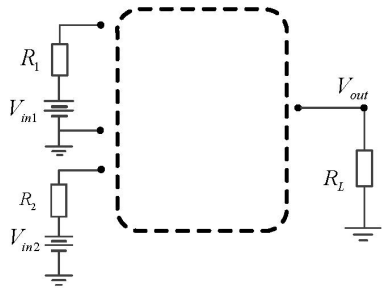


Figure 3.15: The diagram of embryo circuit for temperature sensor circuit. Where $V_{in1} = 15V$, $V_{in2} = 5V$, $R_1 = 1K\Omega$, $R_2 = 1K\Omega$, load resistor $R_L = 10K\Omega$, and the output voltage V_{out} will be measured to be evaluated.

- Fitness Function: At the different circuit temperatures T_i, the output voltage V_{out} is

different, which will be measured to be evaluated. The *i-th* target value of output voltage is linear-related to the circuit temperature T_i, which is defined as $V^*_{outi} = \eta T_i$. η is a constant representing the linear relation between circuit temperature and output voltage. The fitness function is defined as following [150]:

$$fitness = -\sum_i \left(V_{outi} - V^*_{outi}\right)^2. \tag{3.11}$$

- Device Set: The devices applied to evolving temperature sensor circuit are the same as those used for evolving voltage reference circuit mentioned in Section 3.3.1.

- Circuit Quality Measurement: Different from voltage reference circuit, the standard of "Hit" for temperature sensor circuit is that the absolute difference between the measured output voltage and target ($error$) is less than or equal to $0.1V$, which is the same as the one has been set in [37]. The rate of "Hit" will indicate the evolved circuit quality.

Experimental Setup for Gaussian Function Generator

The final task is to evolve a Gaussian function generator, of which output current is a Gaussian function of input voltage. All of the candidate circuits will be simulated with DC sweep, where the variation range of input voltage is $2V \leq V_{in1} \leq 3V$. The target output current is the Gaussian function with a peak value $I^{max}_{out} = 80nA$ in correspondence of $V_{in1} = 2.5V$ and sweep step is $10mV$, which provides a total of 101 measured points for DC sweep simulation.

- Embryo Circuit: The embryo circuit for evolving Gaussian function generator is shown in Figure 3.16 There are four accessible nodes, where one is the variable voltage supply (V_{in1} with series resistor R_1), one is the fixed voltage supply (V_{in2}), one is the output

terminal (with another voltage supply V_L), and the final one is ground. The part marked by the dashed line will be further evolved.

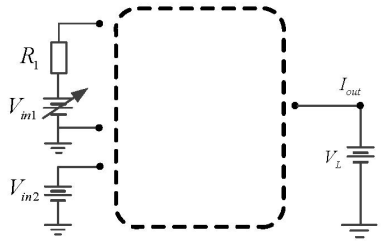


Figure 3.16: The diagram of embryo circuit for Gaussian function generator. Where $2V \leq V_{in1} \leq 3V$, $V_{in2} = 5V$, $R_1 = 1\Omega$, and $V_L = 2.5V$. I_{out} will be measured to be evaluated.

- Fitness Function: During the circuit evolution, the output current will be measured to be evaluated. There will be the different target values of output current corresponding to the different input voltage V_{in1}. Therefore, the fitness function is defined as follows [150]:

$$fitness = -10^{14} * \sum_{i} \left(I_{outi} - I^*_{outi}\right)^2 . \tag{3.12}$$

  where I_{outi} is the measured current and I^*_{outi} is the corresponding target value. The value -10^{14} is a factor to normalize the square of the error, since the unit of swept current is nano-level, and the square of the differences between the target current and measured current will be much smaller.

- Device Set: MOSFETs are applied to evolving Gaussian function generator and resistors are also necessary for Gaussian function generator. The type and model of the devices used in this work are the same as the baseline approaches [190] and [37].

- Circuit Quality Measurement: As for the Gaussian function generator, there is also minor change for the standard of "Hit". The absolute difference between measured current I_{outi} and target value I^*_{outi} $(error)$ is less than or equal to $5nA$, a "Hit" will be scored. The circuit quality measurement applied in this work is the same as the one used in [37].

Experimental Setup for Memristor-based Pulse Generation Circuit

In order to evolve a memristor-based pulse generation circuit, the selection of memristor models is the first consideration. Since the invention of HP TiO_2 memristor [281], some research groups pay their attention to modelling memristor dynamic behaviors into formulas [311] [21]. HP model is a classic memristor device model introduced by Strukov et. al [281], where a time-domain differential equation was proposed to describe the physical behavior of linear ion drift in a memristor [165]. Therefore, in this work, Hewlett-Packard (HP) TiO_2 memristor model is adopted for the concept verification in our design and simulation.

Since the invention of HP TiO_2 memristor [281], some research groups pay their attention to modelling memristor dynamic behaviors into formulas. Volt ampere characteristics of memristor can be described by the following algebraic equation [281]:

$$v(t) = [R_{on}\frac{w(t)}{D} + R_{off}(1 - \frac{w(t)}{D})]i(t), \tag{3.13}$$

where the total length of memristor is denoted by D; the resistances of oxygen vacancy and oxygen-deficient vacancy parts of memristor are represented by R_{on} and R_{off}, respectively; $w(t)$ is the oxygen vacancy part; and d$v(t)$ and $i(t)$ are the voltage applied on memristor and the current flowing across the memristor, respectively.

The state variable of memristor changes with the applied external signal, which can be described by the following differential equation:

$$\frac{dx(t)}{dt} = \frac{\mu_v}{D^2}i(t)f(x(t)), \tag{3.14}$$

where μ_v is the dopant mobility of the material and $x(t)$, which is regarded as the state variable of memristor, is the proportion of the length of the oxygen vacancy part $w(t)$ and

Table 3.1: The parameter setting of proposed GP algorithm

Parameters	Value
Number of generations	800
Population size	100
Crossover rate	0.6
Mutation rate	0.2
Value crossover rate	0.6
Tree depth	4-7
Tournament size	20

total length of memristor D. $f(x(t))$ is a window function to get over the boundary effect of memristor [133], which can be described as follows:

$$f(x(t)) = 1 - (2x(t) - 1)^{2p}, \tag{3.15}$$

where $p \in Z_+$ is a parameter of controlling the non-linear degree of the window function.

In order to evolve the pulse generation circuit, a proper embryo circuit should be predefined. As shown in Figure 3.17, five circuit components are predefined in the netlist, which are input source V_{in}, pull-up voltage V_{cc}, pull-off voltage $-V_{cc}$, ground port and the load R_{load}, respectively. And their position information (0, 1, 2, 3 ,4) will be added into the terminal node set in advance.

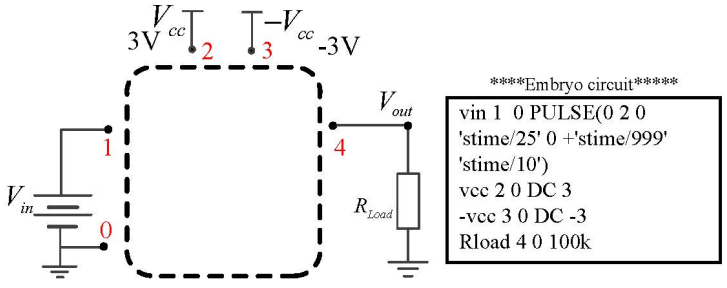


Figure 3.17: Schematic diagram and netlist of embryo circuit for evolving memristor-based pulse generation circuit.

Besides setting up the embryo circuit, the following items also need to be set up:

- Input and target output: Similar with the neuronal dynamic behaviors, the output pulse of pulse generator can be triggered by accumulated potential. Therefore, the accumulated potential will be set as the input of the evolved circuit. For simplification, the saw-tooth wave with $2V$ amplitude and $1ns$ period will be used as input in this work. The square pulse is the most commonly generated by pulse generation circuits, and will thus be set as the target output, of which the amplitude is $2V$ and period is $1\ ns$.

- Set of Devices: To our best knowledge, there is no pulse generation circuit that is only composed of memristors, even in the logical circuit content [356]. Memristors and MOSFET are commonly used for constructing pulse generation circuits [362] [323]. Therefore, in our design, in order to manipulate the dynamic switching behaviors of memristor sufficiently, MOSFET, resistors, and memristors are added into the devices set.

- Simulation Options: Combined with the rate of state switching of memristor, the simulation time is set as $10ns$. To trade off the execution time and sample accuracy, the number of sample points of the target output of the circuit is set as 1000. We need the variation of output with time; therefore, the simulation type is the transient analysis.

Besides the circuit configuration, the parameters of the algorithms were chosen based on preliminary experiments, and are listed in Table 3.1. The fitness function of evolving the memristor-based pulse generation circuit is defined as follows:

$$fitness = -100 * \frac{\sum_i |V_{outi} - V^*_{outi}|}{N}, \tag{3.16}$$

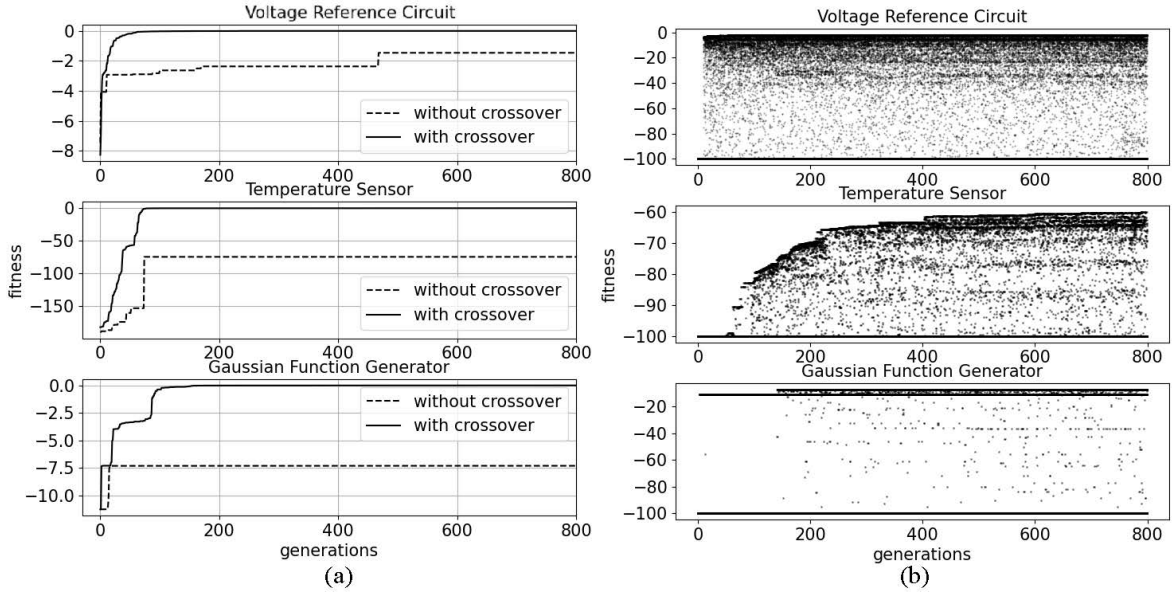


Figure 3.18: (a) The average fitness comparisons with crossover and without crossover operation for different benchmark circuits. (b) Fitness distribution without structure check for different benchmark circuit.

where N is the total number of sampled points, which is set as 1000; V_{outi} is the *i-th* sampled point of output voltage and V^*_{outi} is its corresponding target value.

3.3.2 Validation of Crossover Operators and Feasibility checks

As explained in Section 3.1.3, our tree-based representation lends itself to a suitable crossover operator. In particular, our topology crossover stands for exchanging sub-circuits between individuals. This may lead to better fitness than algorithmic designs that do not make use of crossover, being an advantage over algorithm designs such as graph-based designs. Therefore, we evaluate whether the introduction of our crossover operators is helpful to improve fitness. Figure 3.18 (a) shows the fitness across generations when applying and when not applying the crossover operators. We can see that the evolution with crossover operators leads to better fitness results than the one without crossover operators, confirming the benefits of our tree-based representation, which lends itself to the adoption of crossover operators.

In addition, the effectiveness of the structure check proposed in Section 3.1.2 is verified by executing the experiments without the structure check, where the infeasible individuals are given a penalty instead of applying the proposed structure check to revise them. Several approaches in the literature are based on penalties, such as Gan's work [78]. As Figure 3.18 (b) shows, there are a lot of infeasible individuals ($fitness = -100$) in the population, which may limit the diversity of the population incurring worse fitness.

3.3.3 Comparison Under Different FEs

This section gives the fitness evaluation results for each benchmark circuit. Several metrics are used for a statistical evaluation of the proposed method, such as BF and MBF, which have been widely used to measure the performance of evolutionary algorithms [69, 37]. BF defines the best fitness obtained in 20 runs and MBF is the acronym of Mean Best Fitness, which is the average value of the best fitness obtained in each run. Table 3.2 shows the results for the experiments with different number of evaluations. As Table 3.2 shows, with the increase of the number of fitness evaluations from 2×10^4 to 8×10^4, BF and MBF are more closing to 0, as expected. The average fitness and error curves for 20 runs of the experiments of each benchmark circuit will be given in the appendix.

3.3.4 Comparison With Other Existing Approaches

To further validate the proposed method, this section compares its results with those of the previous approaches, which are shown in Table 3.3. The data shown in the last column of Table 3.3 are the average results for 20 runs of the proposed method. As mentioned in [190], despite the differences in the number of runs in other's work, it is useful to consider side by side the results obtained with those methods.

Table 3.2: Results for the experiments with different the number of evaluations

Tasks	Evaluations	BF*	MBF±std
Voltage reference	2×10^4	-0.0228	-0.8141±1.0961
	5×10^4	-0.0081	-0.2594±0.7815
	8×10^4	-0.0070	-0.0319±0.0231
Temperature sensor	2×10^4	-0.0093	-0.1941±0.2058
	5×10^4	-0.0089	-0.0502±0.0480
	8×10^4	-0.0084	-0.0233±0.0232
Gaussian function	2×10^4	-0.0158	-0.2865±0.2472
	5×10^4	-0.0074	-0.1104±0.0944
	8×10^4	-0.0063	-0.0883±0.0862

* Fitness is with negative sign, so the best fitness (BF) is to maximize the negative fitness value.

Table 3.3: Comparisons with previous work for the three benchmark circuits

Parameters	Koza's [150]	Matthiussi's [190]	Federico's [37]	**Ours**
Vol. reference				
Absolute value of fitness	6.6	2.64	0.112	**0.0233**
Evaluations	5.12×10^7	5.6×10^6	1.86×10^6	$\mathbf{8\times10^4}$
Hits/max	89.9/105	98.1/105	70.7/105	94/105
Components	67	70.2	32	**15**
Tem. sensor				
Absolute value of fitness	26.4	1.13	0.065	0.0883
Evaluations	1.6×10^7	6.5×10^6	6.14×10^6	$\mathbf{8\times10^4}$
Hits/max	16/21	20.3/21	19.9/21	19.1/21
Components	54	27.8	33	**21**
Gau. function				
Absolute value of fitness	0.094	0.3	0.036	**0.0319**
Evaluations	2.3×10^7	4.3×10^6	6.23×10^6	$\mathbf{8\times10^4}$
Hits/max	101/101	98.3/101	85.0/101	**99.6/101**
Components	14	36	28	30

As for the results of evolving voltage reference circuit, the proposed method produces better fitness (0.0233) with less number of evaluations (8×10^4), where the fitness of other work are 6.6 (Koza's), 2.64 (Mattiusssi's) and 0.112 (Federico's), respectively. It is worth noting that the fitness of the proposed work is much better than that of Koza's [150] although the two methods are all based on tree structure. This highlights the advantage of our proposed tree representation. Hit rate of the proposed method is greater than the work presented by Koza [150] and Federico [37]. Moreover, our evolved voltage reference circuit has fewer components than all others.

Table 3.4: Comparisons of manual-designed circuits with evolved circuits

	Work	Implementation	#components
Vol. reference	ours	BJT+resistor	**15**
	Manual design[31]	BJT+resistor+capacitor	29
Tem. sensor	ours	BJT+resistor	**21**
	Manual design [191]	BJT+resistor+diode	23
Gau. function	ours	MOSFET+resistor	**30**
	Manual design [228]	MOSFET	30

In the case of temperature sensor circuit, the fitness produced by Federico's method is slightly better than the one produced by our method, but using about two orders of magnitude more circuit evaluations. Moreover, the result temperature sensor circuit evolved by our method contains the least number of components compared with all other methods.

In the case of Gaussian function generator, the proposed method also can produce better fitness (0.0319) with less number of evaluations (8×10^4). In addition, the hit rate of proposed method is higher than other methods. And also the evolved results of temperature sensor circuit have fewer components, which is more compact than the left three. And the hit rate of proposed method is also competitive with other works. The circuit evolved by our proposed method is more compact than the one proposed by Matthiussi [190].

As it can be seen in the above-mentioned experiment results, the feasibility of our proposed approach has been verified considering the evolved circuit performance and the fitness comparisons with previous work. In addition, our method usually improves the fitness results while using less number of evaluations, achieves a competitive hit rate, and usually uses fewer components to construct the circuits.

As for the manual design of the benchmark circuits, engineers have proposed different ways to construct these circuits, while the strong experience and circuit knowledge are highly required. Some classic manual-designed circuits for benchmark circuits are applied to compare with those of our approach evolved. Paul et. al [31] proposed a voltage reference

circuit, which is composed of 15 BJTs, 13 resistors and 1 capacitor. Meijer et. al [191] proposed a temperature sensor, which is composed of 8 BJTs, 7 diodes, 2 capacitors and 6 resistors. Popa [228] proposed a Gaussian function generator that consists of 30 MOSFETs. The comparison with manual-designed circuits is listed in Table 3.4. Compared with these manual designs, our proposed method provides an automated design tool for designing the analog circuits without the high requirement of circuit experience and knowledge, and the evolved circuits are human-competitive, which not only can realize their corresponding functions but also have compact size.

We have also inferred that why our proposed method outperformed other's work is fused by two parts. Firstly, sub-circuits of a whole circuit could be represented by the branches of a tree, therefore, the crossover operation executed on the tree stands for the exchanging between sub-circuits, which is beneficial for the evolution process as shown in Section 3.3.2. In Koza's tree [152], the sub-circuits are determined by not only the corresponding branches but also the other function nodes located in its parent's node level. As a result, their crossover operations between the branches may miss the corresponding function nodes in its parent's node level, which will be disruptive to evolution process. Secondly, because of three types of structure check for circuit, each circuit individual decoded by our tree-based representation could be simulated and evaluated, preventing the problem of infeasible chromosomes in Federico's approach [37]. In Federico's work [37], the circuit individual that cannot be simulated could be still generated, though they will are strongly penalized. As shown in Section 3.3.2, the structural checks adopted by our approach are more efficient than the use of a penalty. In addition, according to circuit simulation results, our evolved circuits can ensure the circuit feasibility successfully.

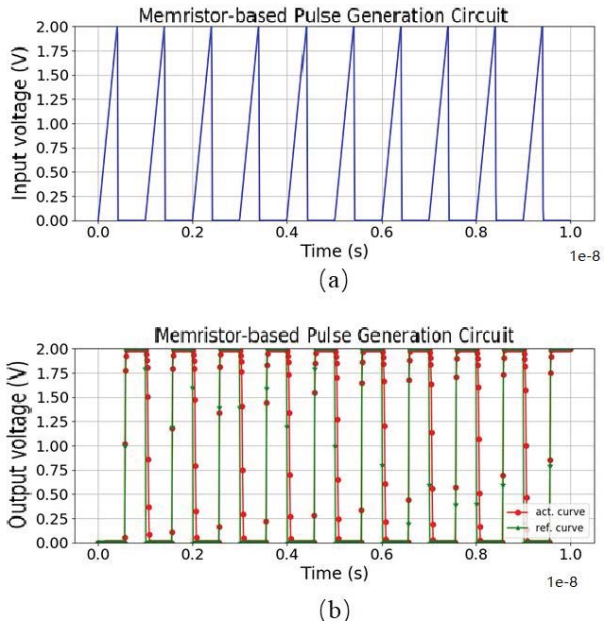


Figure 3.19: Saw-tooth wave input and the circuit simulation results of the best evolved circuit. (a) Saw-tooth wave input. (b) Target voltage

Table 3.5: Area and value range of the devices

Devices	Memristor	MOSFET*	Resistor
Area	$9nm^2$	$0.6075\mu m^2$	$2025nm^2$
Value range	OFF or ON	/	200-200K

* The area of one MOSFET is calculated by Equation 3.19.

3.3.5 Evolution of a Memristor-based Pulse Generation Circuit

The results for the experiments with a different number of evaluations are shown in Table 3.6. As it shown in Table 3.6, the BF (Best fitness for one run) and MBF (Mean Best Fitness) across the 20 runs improve with the increase of the evaluations. Figure 3.21 shows the average fitness and error curves for 20 runs of the experiments.

Figure 3.19 shows the input, output and target voltage of the evolved memristor-based pulse generation circuit. As we can see, under the triangle input voltage, the evolved circuit can generate a regular pulse, which is highly matched to the target curve. In order to qualify the output performance of the evolved pulse generation circuit, Mean Square Error

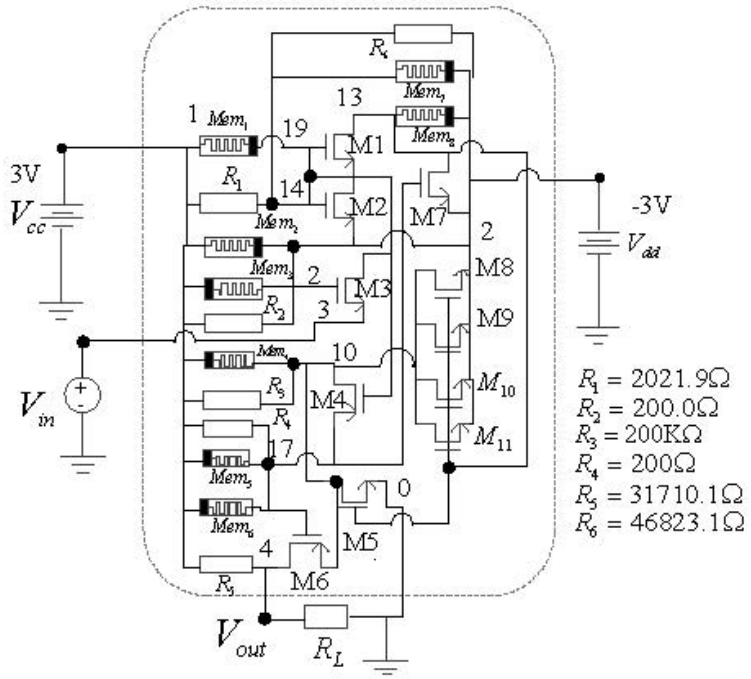


Figure 3.20: The best evolved results of voltage reference circuit.

(MSE), which has been widely applied for evaluating the differences between the actual and target output of circuit [12], is applied for measuring the quality of the circuit output in this work.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(V_{outi}^{*} - V_{outi}\right)^{2}, \tag{3.17}$$

where V_{outi}^{*}, V_{outi} are target voltage value, actual output voltage value for a given data point i, respectively. N is the number of sampled points of the output voltage. MSE of the result shown in Figure 3.19 is 0.15, which meets the requirement of the circuit with similar purpose proposed in [265]. Figure 3.20 shows the evolved memristor-based pulse generation circuit, including the evolved resistance of the resistors contained in the circuit. The circuit contains 8 memristors, 6 resistors and 10 transistors. The initial state of all the memristors in the evolved circuit is OFF.

Table 3.7 gives the comparisons of the evolved circuit and other manual design circuits with similar purpose. To validate the proposed approach, we compare the works from

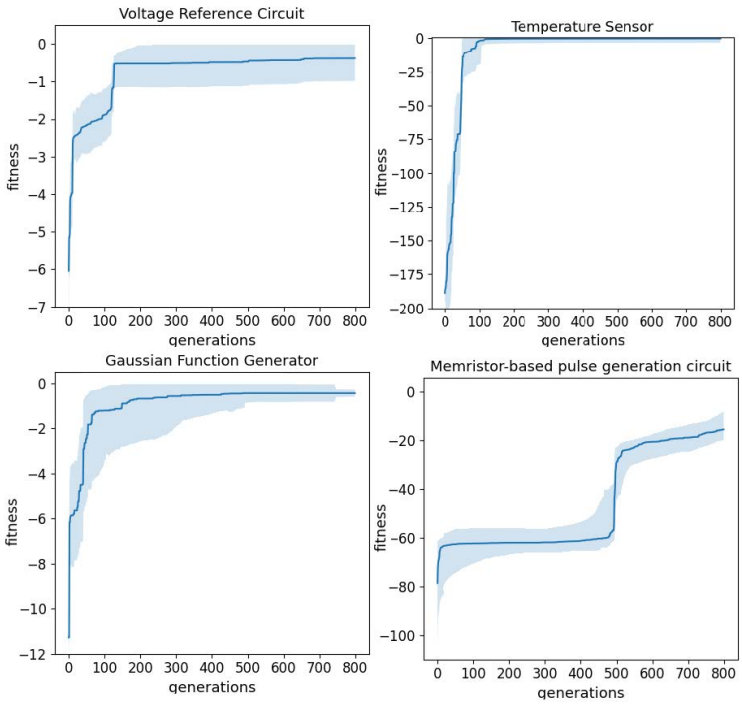


Figure 3.21: The average fitness for 20 runs of the experiment on different tasks.

different perspectives, which are the usage of external chip, the ways of realizing, the number of transistors and capacitors, area and energy. According to Wu et al. [330], the energy dissipation per pulse can be defined as the differences between the input power consumption P_{IN} and output power consumption P_{OUT} during the duration of one pulse, where P_{IN} and P_{OUT} can be calculated by the following equation:

$$P = \int_0^T |V(t) \cdot I(t)| dt. \tag{3.18}$$

Where $V(t)$ and $I(t)$ are the voltage and current at the sampled points.

In addition, the area of the single MOSFET can be calculated by the following equations:

$$AREA_{MOS} = W \times L \times k. \tag{3.19}$$

where W and L refer to the width and length of the channel, and k is a factor showing how

Table 3.6: Results for evolving memristor-based pulse generation circuits with different the number of evaluations

#Evaluations	BF*	MBF±std
2×10^4	-16.84	-37.32±16.54
4×10^4	-12.45	-31.28±19.02
6×10^4	-11.46	-22.01±12.23
8×10^4	-10.69	-15.02±4.54
10×10^5	-10.47	-12.91±2.10

much bigger a transistor is than its channel area. As for the area of one HP memristor, it is set as 9 nm^2, as proposed in previous work [281]. Considering the compatibility of memristor and MOSFET, 45nm CMOS technology is applied in this work for MOSFET and resistor. The area of one resistor is set as $2025nm^2$ ($45nm \times 45nm$). The area of one MOSFET is set as $0.6075\mu m^2$($450nm \times 450nm \times 3$). Our current estimation of power and area of circuits may not be accurate and could be improved in future work. Table 3.5 shows the area and device value range of the devices to be used in evolving circuits. The initial states (ON or OFF) of different memristors and the specific value of different resistors will be determined by the proposed algorithm.

As shown in Table 3.7, some researchers applied an external chip to generate pulse, 555 timer, which contains 25 transistors and 2 capacitors, leading to large circuit area and power consumption [265] and [322]. Petit et al. used 19 transistors and 3 capacitors to implement the pulse generation, which occupies $18340\mu m^2$ circuit area and generates $4500nJ$ energy for one pulse [27]. In the work of Wijekoon et al. [328], 20 transistors and 1 capacitor were applied to design the circuit for generating pulse, of which area is $1705\mu m^2$ and energy dissipated per pulse is $2.85nJ$.

Compared with the work of [265] and [322], our evolutionary approach prevents applying the external chips as pulse generator, which solves the problems of bulk auxiliary circuits and inefficient resource utilization of chips. Compared with the manual design cir-

Table 3.7: Comparisons of the evolved memristive pulse generation circuit and other works

Works	External chip?	Realizing method	#Transistors	#Capacitors	Area	Energy/pulse
[265]	YES	555 timer	25	2	$9mm^2$	$240nJ$
[322]	YES	555 timer	25	2	$9mm^2$	$240nJ$
[27]	NO	Transistor+Capacitor	19	3	$18340\mu m^2$	$4500nJ$
[123]	NO	Transistor+Capacitor	20	1	$1705\mu m^2$	$2.85nJ$
[362]	NO	Transistor+Memristor	16	1	$23.24\mu m^2$	$125nJ$
[323]	NO	Transistor+Memristor	25	0	$46.37\mu m^2$	-
Ours	NO	Transistor+Memristor	**10**	**0**	$\mathbf{6.08\mu m^2}$	$\mathbf{1.53nJ}$

cuits proposed by [27] and [328], the evolved memristor-based pulse generation circuit is equipped with fewer transistors (0 capacitor), lower energy consumption, and more compact design, which is better for neuromorphic computing. The work proposed in [362] and [323] also applied memristors to implement pulse generation. As for work [362], additional 16 transistors and 1 capacitor were used. As for work [323], 25 MOSFETS are applied extra besides the memristors, which were more than the ones evolved by our proposed method (10 transistors and 0 capacitor).

3.4 Experimental Studies for Evolving Analog Circuit Using Shapley Value-based GP

This section presents the experimental studies of our proposed approach of evolving analog circuit wsing Shapley value-based GP. Specifically, Section 3.4.1 introduces the experimental setup, Section 3.4.2 presents the comparisons of different ablated components of algorithm, Section 3.4.3 gives the comparison with other existing approaches. Section 3.4.4 analyzes how the Shapley value influences the evolution.

3.4.1 Experimental Setup

The proposed method is evaluated on three benchmarks, namely voltage reference circuit, temperature sensor circuit, and Gaussian function generator. These benchmark circuits are widely applied to evaluate the evolutionary analog circuit design [150, 190, 37]. The objective to evaluate the proposed approach against existing ones in terms of minimizing differences between the actual output of the circuit and their corresponding target, and to determine which of its components are most useful for getting desired output. The embryo circuit setup of the three benchmark circuits is the same as the one introduced in Section 3.3.1.

Moreover, memristor research is attracting increasing attention in the field of electronic technology for its nonvolatility, nano-size and energy efficiency [281]. Generally, memristive circuits have more smaller area compared with traditional device-based circuits. However, manipulating memristors to construct the analog circuits with different function is a challenging task. Therefore, we also apply our proposed method for evolving the memristive circuits. The experiment setup for evolutionary circuit design is divided into two parts, which are embryo circuit setup and evolutionary algorithm setup. The specific settings are in Table 3.8. BJT models applied in our work are BC846B for NPN and BC856B for PNP; MOSFET models applied in our work are NMOS_VAR_CH_W for NMOS and PMOS_VAR_CH_W for PMOS. Memristor model applied in our work is based on a AgInSbTe memristor, which is given in Equation 3.20 to Equation 3.22.

Memristive Square Calculation Circuit

This task is to evolve a memristive square calculation circuit, whose output voltage is the square of the input voltage. As for the tasks of evolving memristive circuits in our work, we apply a voltage-controlled threshold memristor model, which is capable of fitting AgInSbTe

memristor [348], and it can be described as:

$$\frac{dx(t)}{dt} = \begin{cases} k_{on} \cdot v(t) \cdot f(x), & \text{if } v(t) > v_{on} > 0; \\ 0, & \text{if } v_{off} \leq v(t) \leq v_{on} \\ k_{off} \cdot v(t) \cdot f(x), & \text{if } v(t) < v_{off} < 0 \end{cases} \tag{3.20}$$

$$f(x) = \begin{cases} (a \cdot (1-x))^p, & \text{if } v(t) > 0 \\ (a \cdot x)^p, & \text{if } v(t) < 0 \end{cases} \tag{3.21}$$

$$v(t) = [R_{on}x + R_{off}(1-x)] \tag{3.22}$$

where state variable x is a normalized width of the conducting layer, whose derivative is matched to memristor current-voltage data; $f(x)$ is a speed adaptive state variable function; v_{on} and v_{off} represent the positive and negative voltage threshold. a, p are scaling parameters, which determine the indirect effect drift speed. R_{on} and R_{off} represent the bounds of device resistance. The fitness function of evolving the memristive square calculation circuit is:

$$fitness = -\sum_i \left(V_{outi} - V^*_{outi}\right)^2. \tag{3.23}$$

Memristive Cube Calculation Circuit

This task is to evolve a cubing circuit, whose output voltage is the cube of the input voltage. Specific setting of circuit simulation and algorithm are given in Table 3.8 and 3.9. The fitness function is also the same as Equation 3.23.

Table 3.8: Embryo circuit setup and algorithm parameters setting

Benchmarks	Embryo Circuit									Algorithm parameter setting	
	Input 1		Input 2		Output		GND	Simulation type	Device set of evolved circuit		
	R_{in1}	V_{in1}	R_{in2}	V_{in2}	V_L	R_L				Parameters	Value
Ref. Vol.	$1k\Omega$	5V	-	-	-	$10k\Omega$	✓	DC sweep	BTJ*, resistor	Population size	100
Tem. Sen.	$1k\Omega$	15V	1K	5V	-	$10k\Omega$	✓	DC sweep	BTJ, resistor	Tournament size	20
Gau. Gen.	1Ω	2.5V	-	5V	2.5V	-	✓	DC sweep	MOS, resistor	Max_iteration	500
Mem. Application	R_{in}	V_{in}	V_{cc}	V_{dd}	V_L	R_L	GND	Simulation type	Devive set of evolved circuit	Shapley_iteration	250
Mem. Square	$1k\Omega$	1V	5V	-5V	-	$1k\Omega$	✓	DC sweep	MOS.*, res., mem.*	P_{cross}	0.8
Mem. Cube	$1k\Omega$	1V	5V	-5V	-	$1k\Omega$	✓	DC sweep	MOS., res., mem.	P_{mutate}	0.2
Mem. Pulse	-	Saw tooth	3V	-3V	-	$100k\Omega$	✓	Tran. ana.	MOS., res., mem.	$P_{valuecross}$	0.2

Table 3.9: The parameters of circuit simulations and sampling

	Output signal	Sweep range	# Sampled point
Ref. Vol.	V_{out}	4-6V; 0-100 °C	105
Tem. Sen.	V_{out}	2-3V; 0-100 °C	21
Gau. Gen.	I_{out}	2-3V	101
Mem. Square	V_{out}	-0.25-0.25V	21
Mem. Cube	V_{out}	-0.25-0.25V	21
Mem. Pulse	V_{out}	0-10ns	1000

Memristive Pulse Generation Circuit

This task is to evolve a pulse generation circuit, where the input is the saw-tooth wave voltage and the output is the pulse voltage. Considering the amplitude of target pulse, the fitness function of evolving the memristor-based pulse generation circuit is defined as follows:

$$fitness = -100 * \frac{\sum_i |V_{outi} - V^*_{outi}|}{N}, \tag{3.24}$$

where N is the total number of sampled points, which is set as 1000; V_{outi} is the *i-th* sampled point of output voltage and V^*_{outi} is its corresponding target value.

3.4.2 Comparison of Different Ablated Components of Algorithm

In this section, we perform an ablation study of Shapley circuit tree method on the stage 2 of the evolution. Specifically, the Shapley circuit tree method on the stage 2 is ablated, remaining the same as stage 1. Figure 3.22 (top row) shows the fitness comparisons with and without Shapley on stage 2 for some representative circuits. All the individuals generated in the 250-th generation will be reserved, and then two different evolution methods will be applied to these individuals, which are with and without Shapley method, respectively. According to the fitness comparisons after 250 generations, the fitness based on the evolution with Shapley values can converge to better fitness faster than the one without Shapley. The absolute best fitness ($|BF|$) and their average best fitness ($|MBF|$) for different tasks with and without Shapley are given in Table 3.11. As we can see, within Max_iter.=500, the evolution with Shapley can obtain better fitness compared with the one without Shapley. Combined with stimulating the diversity during the first stage evolution, the Shapley-oriented evolution will guide the genetic operations towards more promising search space, which may accelerate the process of pursuing the better individual.

Moreover, we also compared with our proposed method with other previous evolutionary analog circuit design work [150, 190] on three benchmarks, where the comparisons are shown in Table 3.11. In terms of the fitness results and the number of evaluations, our proposed work achieves better fitness with less number of evaluations.

3.4.3 Comparison with Other Existing Approaches

Besides the fitness results during the evolution, we are also concerned with the size of the evolved circuits. To analyze if our Shapley-oriented stage is beneficial to evolve more parsimonious circuit, we also monitor the circuit area of the individual with the best fitness in

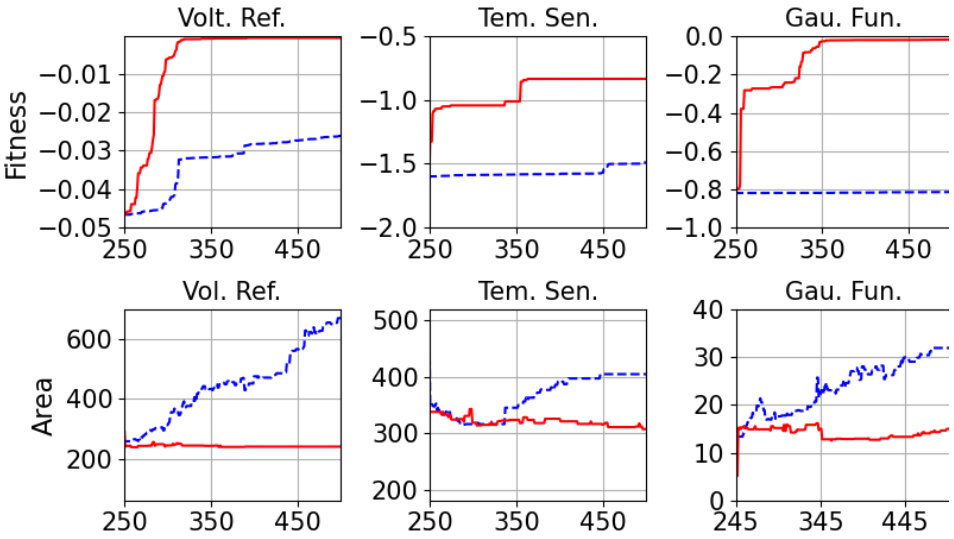


Figure 3.22: Average fitness results and circuit area comparison with (red solid line) and without (blue dashed line) Shapley.

each generation. The circuit size for each device is given in Table 3.5. The average circuit area of the evolved circuit from the ablation study with and without Shapley-oriented stage evolution is given in the Table 3.10. As we can see the circuit area of the circuit evolved with the Shapley-oriented stage is smaller than that of without Shapley-oriented stage. The visualized curve of the circuit area changed during the evolution is shown in Figure 3.22 (bottom row). We take the voltage reference circuit as an example. As for the evolved results with Shapley-oriented stage, during the generations from 250 to 350, the fitness is increasing while its corresponding circuit area has no obvious increase, and during the generation from 350 to 500, even if the fitness have not been witnessed an obvious increase, the area still did not increase, which is different from the blue dash line. As for the evolved results without the Shapley-oriented stage, although the fitness is increasing gradually during the generation from 250 to 500, the corresponding area is growing dramatically.

The circuit area is also compared with that of the previous evolutionary circuit design work [150] [190], which is shown in Table 3.11. Although previous work did not give the circuit area of their evolved circuits, the number of components they have applied is accessible.

Table 3.10: Average fitness and area of evolved circuit with and without Shapley, calculated across 10 runs (Max_iter.=500).

	Shapley?	$\|BF\|$	$\|MBF\|*$	Circuit area
Reference voltage	Yes	0.0005	0.0054	$240.27\mu^2m$
	No	0.0051	0.0261	671.2 μ^2m
Temperature sensor	Yes	0.0099	0.0519	$307.78\mu^2m$
	No	0.0189	0.3981	$401.24\mu^2m$
Gaussian function generator	Yes	0.0096	0.0147	15.01 μ^2m
	No	0.0874	0.4046	32.5 μ^2m
Memristive pulse generator	Yes	9.3417	12.7412	2.78 μ^2m
	No	10.8246	22.4984	5.24 μ^2m
Memristive Squaring circuit	Yes	0.5871	2.2559	2.60 μ^2m
	No	2.1469	5.4223	5.99 μ^2m
Memristive cubing circuit	Yes	0.2411	2.3842	4.14 μ^2m
	No	1.2861	4.8317	7.14 μ^2m
P-value (with Shapley VS without shapley)=0.1892				

* $|MBF|$ is the absolute value of the mean value of best fitness cross 10 runs.

Except the Gaussian generator task, our work outperforms the existing works over other two benchmarks in terms of the the number of components.

The results for the evolution of the memristive circuits are in Table 3.12, where we compare our approach against existing work that manually designed circuits for the same benchmarks [323, 27, 58, 1, 245]. The fitness was not provided in those existing works, as they were manually designed. However, the output voltage achieved by our evolved circuits was similar to the target output voltage. An example is in Figure 3.23(e). In terms of the circuit area, the evolved circuit by our proposed method outperformed existing work. Not all manual designs made use of memristors. However, the area and number of components used by our evolved circuit were smaller even when compared against [323], where the manually designed circuit had access to the same types of device as our evolutionary framework (MOS and memristor).

Table 3.11: Comparisons with previous work for the three benchmark circuits

Parameters	[150]	[190]	[263]*	Ours
Reference voltage				
Evaluations	5.12×10^7	5.6×10^6	5×10^4	$\mathbf{5{\times}10^4}$
$\|MBF\|$	6.6	2.64	0.112	**0.0054**
#Components	67	70.2	32	**15**
Area	-	-	$293.31\mu^2m$	$\mathbf{240.27}\mu^2m$
Temp. sensor				
Evaluations	1.6×10^7	6.5×10^6	5×10^4	$\mathbf{5{\times}10^4}$
$\|MBF\|$	26.4	1.13	0.065	0.0519
#Components	54	27.8	22	**19**
Area	-	-	$329.24\mu^2m$	$\mathbf{307.78}\mu^2m$
Gau. function				
Evaluations	2.3×10^7	4.3×10^6	5×10^4	$\mathbf{5{\times}10^4}$
$\|MBF\|$	0.094	0.3	0.036	**0.0147**
#Components	14	36	24	25
Area	-	-	$14.98\mu^2m$	$\mathbf{15.01}\mu^2m$

Work [263] is the one proposed in Section 3.2.1.

Table 3.12: Comparisons of our evolved memristive Circuit and other circuits with similar function

	Pulse generator			Squaring circuit			Cubing circuit		
	[323]	[27]	Ours	[58]	[1]	Ours	[245]	[273]	Ours
Devices used	MOS+mem.	MOS.+Cap.	MOS+mem.	BJT+res.	MOS.+res.	MOS+mem.	BJT+res.	MOS.+res.	MOS+mem.
#Compoents	25	19	10	29	13	9	44	41	14
Area	$46.37\mu m^2$	$18340\mu m^2$	$2.78\mu m^2$	$781\mu m^2$	$7.8\mu m^2$	$2.60\mu m^2$	$1810\mu m^2$	$24.6\mu m^2$	$4.14\mu m^2$

3.4.4 Analysis on How the Shapley Value Influences the Evolution

We take the memristive pulse generation circuit as an example to analyze the how the Shapley values influence the circuits evolution. The best evolved circuits in 250-th and 500-th generation and their corresponding voltage are shown in Figure 3.23. According to Figure 3.23(d), we can see that the best evolved circuit in 250-th generation still cannot generate the pulse like target. However, compared with the best evolved circuit in 500-th generation, there are several similar circuit blocks in the one of 250-th generation. Instead of adding more devices and blocks perusing the better fitness, the existing circuit elements with different combination are very promising for the circuit evolution. For example, we already know that an inverter composed by a NMOS and a PMOS will be benefit the circuit

evolution according to Figure 3.23(b). The circuit in 250-th generation also contains an NMOS and a PMOS but with different connection. Shapley-oriented evolution may guide this block to be evolve towards an inverter. Besides the evolved inverter, we also obtain a pair of two PMOS with gates and sources connected. This structure is also useful in the analog circuit design for dividing the current into different branches.

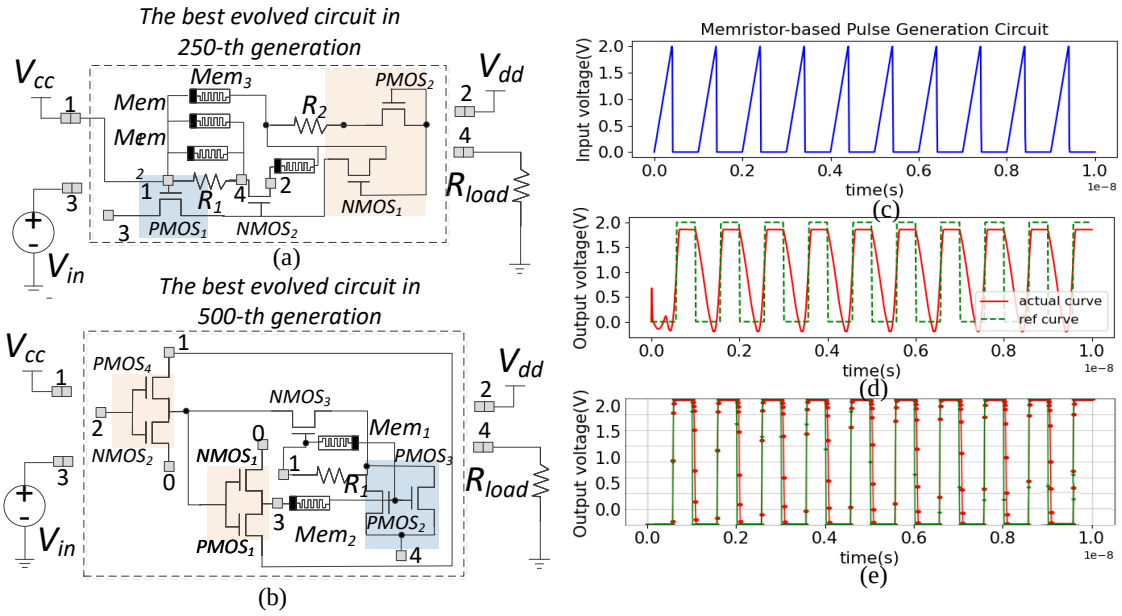


Figure 3.23: The best evolved circuits in 250-th (a) and 500-th (b) generation. The input (c) and output voltages of the best-evolved memristive pulse generator in 250-th generation (d) and 500-th generation (e).

3.5 Summary and Discussion

This chapter introduced analog circuit evolution with improved design ability and evolution process, answering the first research question of this thesis, which is *How to improve general applicability and efficiency of the circuit evolution?*

This chapter answers this research question by proposing a novel tree-based circuit representation and circuit evolution algorithms with improved efficiency. Its contributions

include:

- A novel tree-based representation that can be transformed more efficiently to circuit netlists and is compatible to both of the two-terminal and three-terminal circuits, and for which a suitable crossover operator can be applied, better supporting automated circuit evolution.
- A GP framework that uses the proposed representation and can improve the number of evaluations, fitness and hits over existing literature as shown by our experiments with three benchmark circuits and one memristive pulse generation circuit.
- A case study applying the proposed GP framework to analog memristive circuit design, demonstrating that the proposed framework can lead to a more compact size and greater energy efficiency than existing manually designed circuits.
- A novel evolutionary approach to evolve analog circuits based on Shapley values to identify the contribution of each function node in a circuit-plausible way. The Shapley values are used in the circuit evolution to guide the evolutionary process towards more promising regions of the search space and outperformed existing evolutionary circuit designs and manual design in terms of both fitness and circuit area on our experiments based on three benchmarks and three memristive circuits.
- An analysis of two evolved circuits obtained by different circuit generations, giving a circuit-plausible explanation for the results achieved by our Shapley method.

Chapter Four

Indirect Circuit Representations for Memristive Reconfigurable Architectures[1]

Besides the tree-based circuit representation that encodes the circuit devices and their interconnections introduced in Chapter 3, indirect circuit representation for pre-defined reconfigurable architectures is also necessary [302, 37]. In this thesis, the term "indirect circuit representation" is used to describe a method that encodes a circuit using signals that configure the circuit or other configuration descriptions. This is in contrast to the "direct circuit representation" method, which encodes the circuit based on the schematic that includes devices and their connections. The indirect method offers a different approach to circuit encoding, focusing on configuration signals rather than the physical layout of the circuit. In indirect circuit representations, the circuit structure is represented by a set of encoding

[1]This chapter corresponds to RQ2 in Section 1.1, and is based on our published paper [260] and our paper under review [261]. Specifically, Section 4.2.1 presents the approach proposed in our published paper [260], and Section 4.3 presents the experiment part of our published paper [260]. Section 4.2.1 presents the approach proposed in our paper under review [261], and Section 4.4 presents the experiment part of our paper under review [261]. This footnote, in addition to the statements in Section 1.2, also serves to clarify any detected overlaps between this thesis and my own published papers.

signals [190] or operations [178]. These indirect circuit representations do not correspond directly to the components and connections in the circuit but represent a set of basic operations. By executing these basic operations, circuits with different structures can be generated. Memristive reconfigurable architecture is one type of circuits that is suitable to be encoded by the indirect circuit representation. However, to the best of our knowledge, there are no indirect circuit representations available for memristive EHW with reconfigurable architectures. Given the potential advantages of indirect representation introduced in Chapter 2, such as more compact and more flexible, and since there are various types of memristive reconfigurable architectures, it is worthwhile to explore the indirect circuit representations for them.

In order to design the memristor-based evolvable hardware, the memristive hardware platforms should be designed first, and then the indirect circuit representations will be created to encode the different memristor-based architectures, and further evolve them. Therefore, this chapter aims to answer the following question of the thesis:
RQ2: How to evolve memristive circuits based on reconfigurable architectures?

In the taxonomy proposed by Xia et al. [333], memristor-based reconfigurable architectures are divided into active memristive arrays and passive memristive arrays. The active memristive array refers to memristive arrays where each memristor is connected in series with a transistor. This configuration allows for more precise control over the current flowing through each memristor, which can lead to more diverse dynamic behaviors. A typical example of an active array is the 4T1R memristive architecture, which uses four transistors to control the currents flowing across the memristor. However, this increased control and flexibility come at the cost of greater circuit overhead. On the other hand, the passive memristive array refers to memristive arrays that do not have transistors in the memristor crossbar. This means that there are no transistors to control the current flowing through each memristor. The 1R memristive architecture is a typical example of a passive array.

While this configuration reduces circuit overhead, it also poses greater challenges for control due to issues such as sneak path currents. In this chapter, we propose the evolutionary design of memristive reconfigurable architectures for Reservoir Computing (RC) based on both 4T1R and 1R architectures. By investigating these two typical architectures, we aim to understand their respective strengths and weaknesses and explore their potential in RC applications.

In order to propose the evolutionary design of the memristive reconfigurable architecture for RC, the memristive network for RC is introduced first in Section 4.1. Section 4.2 will then introduce the evolutionary design of the memristive reconfigurable architectures for Reservoir Computing (RC). Section 4.3 presents the experimental studies of 4T1R-based memristive evolutionary design for RC. Section 4.4 presents the experimental studies of 1R-based memristive evolutionary design for RC. The summary of this chapter is given in Section 4.5.

4.1 Memristive Network for RC

Memristive networks can be used in reservoir computing by incorporating memristors into the reservoir layer of the network. Memristors can be used to create a dynamic system in the reservoir layer that can adapt and change over time based on the input data. This adaptive behavior allows the network to learn and respond to complex patterns in the data, making it well-suited for tasks such as speech recognition and time series prediction. This section will first introduce the memristor model that will be used to constructing our memristive reconfigurable architecture for RC in Section 4.1.1, and then the computing model of memristive reservoir computing will be introduced in Section 4.1.2.

4.1.1 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 [288], 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 [45], 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. [45] 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:

$$i = (1-x)\alpha[1-e^{-\beta v}] + x\gamma sinh(\delta v), \tag{4.1}$$

$$\dot{x} = (\lambda[e^{\eta_1 v} - e^{\eta_2 v}] - \frac{x-\theta}{\tau})f(x), \tag{4.2}$$

$$\dot{\varepsilon} = \sigma(e^{\eta_1 v} - e^{\eta_2 v})f(x), \tag{4.3}$$

$$\dot{\tau} = \theta(e^{\eta_1 v} - e^{\eta_2 v}), \tag{4.4}$$

$$f(x) = \frac{(sign(v)+1)(sign(1-x)+1) + (sign(-v)+1)(sign(x)+1)}{4}, \tag{4.5}$$

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

Table 4.1: The parameters of applied memristor model

Model parameters [45]	Meaning	Value
α	Prefactor corresponding to barrier height for Schottky barrier	1e-4
β	Exponent corresponding to depletion width for Schottky barrier	0.2
γ	Prefactor corresponding to barrier height for tunneling	1e-3
δ	Exponent corresponding to effective tunneling distance in the conducting region	1
ε	Retention of the Ohmic-like conducting channel	0.1
η_1	Interface effect with positive voltage	4
η_2	Interface effect with negative voltage	2
λ	Positive constant to control the change rate of x	0.005
τ	Diffusion time	0.5
θ	Positive-valued coefficient for τ	0.01
σ	Positive-valued coefficient for ε	0.0001

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 4.1. They were adapted from the parameters of the bipolar model [45] by tuning through circuit simulation tests to ensure the memristors can generate fading dynamics within our simulation time window (0.5s) one by one.

According to Equation (4.1), 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. Figure 4.1 (a) shows the I-V hysteresis curves of the forgetting memristor model [45], 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 Figure 4.1, where the nonlinearity transformation is also explicit.

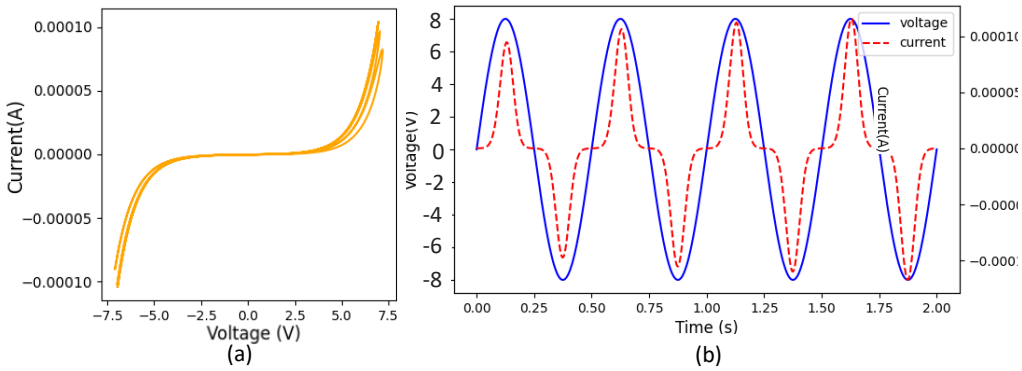


Figure 4.1: (a) I–V hysteresis curves of the forgetting memristor model [45] (b) The circuit simulation result with applied sine voltage.

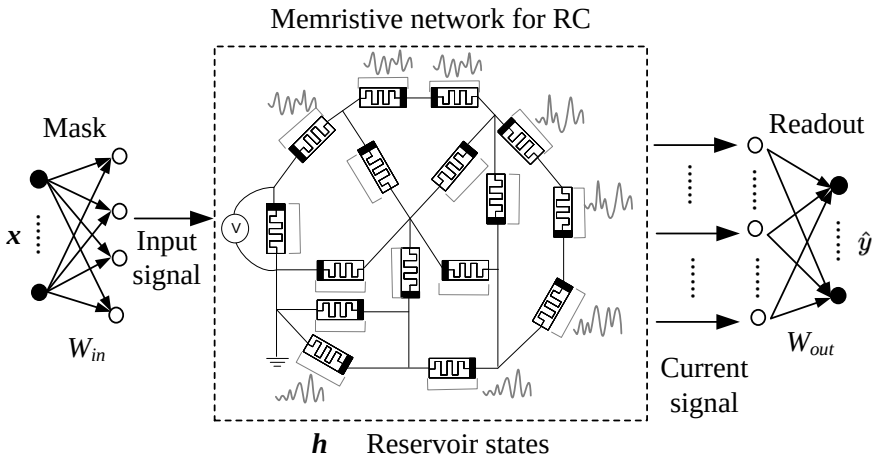


Figure 4.2: Schematic illustration of memristive network for RC with random topology, redrawn from work [286]

4.1.2 Memristive Reservoir Computing Model

Figure 4.2 shows the schematic illustration of the memristive network for RC with random topology [286]. 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 [68, 286]. As for our proposed approach, the same as the previous work [68, 286], 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 desired target output $\mathbf{y}(t) \in \mathbb{R}^{1\times n}, t \in \{1, ..., T^*\}$ 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:

$$\begin{aligned} W_{out} &:= \underset{W\in\mathbb{R}^{N\times n}}{\arg\min} \left(\sum_{t=1}^{T^*} ||\hat{\mathbf{y}}(t) - \mathbf{y}(t)||^2 + \lambda||W||^2 \right) \\ &= \underset{W\in\mathbb{R}^{N\times n}}{\arg\min} \left(\sum_{t=1}^{T^*} ||\mathbf{h}(t) \times W - \mathbf{y}(t)||^2 + \lambda||W||^2 \right) \end{aligned} \tag{4.6}$$

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:

$$W_{out} = (H^\mathsf{T} H + \lambda \mathbb{I}_N)^{-1} H^\mathsf{T} \mathbf{y}. \tag{4.7}$$

As mentioned in Section 4.1.1, a memristor could be generally described by an alge-

braic equation and a differential equation, which are as follows:

$$I = G(w, V)V, \tag{4.8}$$

$$\frac{dw}{dt} = f(w, V), \tag{4.9}$$

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 $\hat{\mathbf{y}}$.

4.2 Evolutionary Design of Memristive Reconfigurable Architecture for RC

Based on the different memristive reconfigurable architectures, the indirect circuit representations for 4T1R-based reconfigurable architecture and 1R-based reconfigurable architecture are proposed in Section 4.2.1 and Section 4.2.2. Based on the indirect circuit representations, their corresponding evolutionary operation design is also given in Section 4.2.1 and Section 4.2.2, respectively. Section 4.2.3 further proposes the main evolutionary algorithm to evolve the memristive reconfigurable architecture for RC.

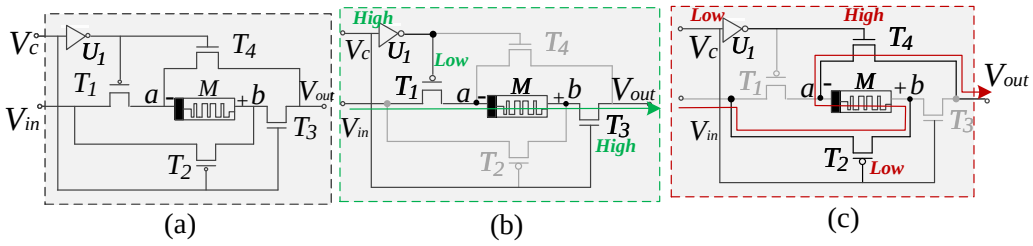


Figure 4.3: (a) Circuit schematic of reconfigurbale memristor-based unit; (b) Equivalent circuit when $V_c = 1$ (High voltage level); (c) Equivalent circuit when $V_c = 0$ (Low voltage level).

4.2.1 Evolving 4T1R-based Reconfigurable Architecture

4T1R-based Memristive Reconfigurable Architecture

We propose to construct the memristive network for RC in a reconfigurable manner based on the 4T1R Reconfigurable Memristive Unit (RMU) from [338] 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 Figure 4.3. In Figure 4.3, V_{in} and V_c represent 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. Figure 4.3 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. Figure 4.3(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. Figure 4.3(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.

Figure 4.4 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 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 Figure 4.5 shows an example circuit composed of 9 RMUs. The

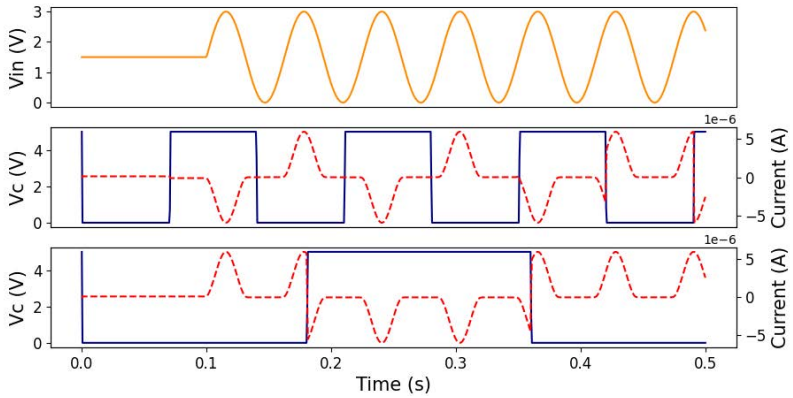
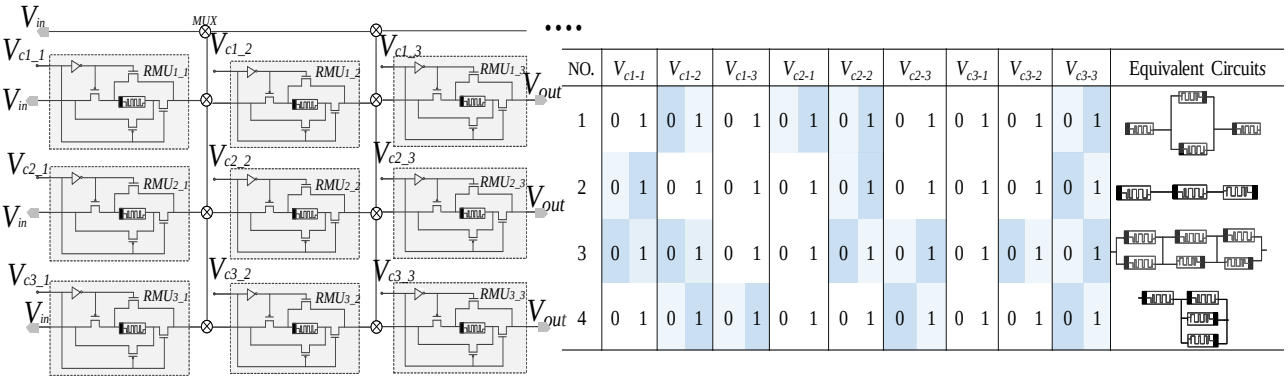


Figure 4.4: Simulation results of reconfigurbale memristor-based unit. The yellow line denotes the V_{in}, the red dash line denotes the current flowing across the memristor and the blue solid line denotes the configuration voltage V_c.

right part of Figure 4.5 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 behaviors 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



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	

Figure 4.5: Circuit example composed by 9 RMUs and the corresponding equivalent circuits under the different states of control voltage V_c

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.

Indirect Circuit Representation for 4T1R-based Reconfigurable Architecture

As mentioned in the previous Section 4.1, 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 (4.7).

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 Figure 4.4, 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

Table 4.2: Genome of proposed 4T1R 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.

control voltage V_c, by which the currents flowing across the memristors can be controlled to present different dynamic behaviours. These are the indirect circuit representation of the 4T1R-based recongfigurable architecture for RC model, using the width of the control signal to represent the connection states instead of the circuit wires.

Table 4.2 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.

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 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) [129], so that the dynamic information flow through the reservoir nodes is not too cluttered. Therefore, we adopt cycle reservoirs with regular jumps (CRJ) [234] to initialize the candidate reservoirs. According to previous work [234], 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 terms of 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. Moreover, *Adapt* operation is also proposed to ensure the sparse topology during the evolution, of which pseduo code is given in Algorithm 4.1. This operation is designed so that the reservoir connections can be evolved gradually and the reservoir is kept sparsely connected [203]. $IsAdapt$ 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

Algorithm 4.1 Pseudo code of adaptive sparse for evolving 4T1R-based reconfigurable architecture $Adapt(genome, g)$

Input: *genomes* in the current generation
Output: *genomes* after $Adapt()$ operations

```
set sparsity ε
if IsAdapt then
    remove a fraction ε of the smallest positive weights by setting their corresponding W_bool^{i,j} to zero
    if g mod e == 0 (at every e generations) then
        return the sparse connection
    else
        add randomly new weights in the same amount as the ones removed previously
    end if
end if
Return genomes after Adapt()
```

to zero, are removed. And then, new weights are added randomly in the same amount as the ones previously removed.

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.
- 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 node 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 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.

Crossover

Algorithm 4.2 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

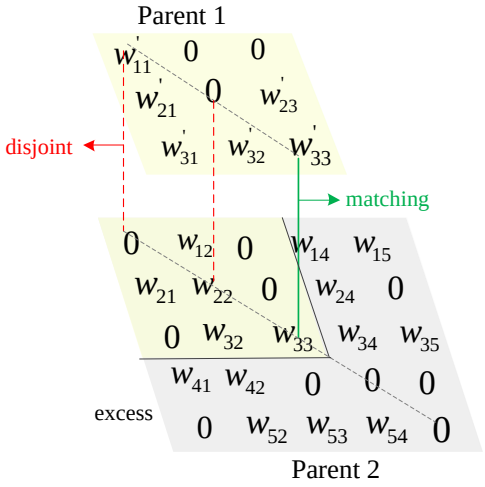


Figure 4.6: Crossover operation for the parents with different sizes.

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 [40]:

$$w_{ij} = \begin{cases} \frac{w_{ij}^1 + w_{ij}^2}{2} & \text{if } w_{ij}^1, w_{ij}^2 \neq 0 \text{ and } random < 0.5, \\ w_{ij}^1 & \text{if } w_{ij}^1, w_{ij}^2 \neq 0 \text{ and } 0.5 \leq random < 0.75, \\ w_{ij}^2 & \text{if } w_{ij}^1, w_{ij}^2 \neq 0 \text{ and } 0.75 \leq random < 1.0, \\ w_{ij}^1 & \text{if } w_{ij}^2, = 0 \text{ and } w_{ij}^1 \neq 0, \\ w_{ij}^2 & \text{if } w_{ij}^1, = 0 \text{ and } w_{ij}^2 \neq 0. \end{cases} \tag{4.10}$$

Figure 4.6 shows an example that explains how the parents with different sizes execute the crossover operation. As shown in Figure 4.6, the sizes of $parent_1$ and $parent_2$ are different, where $parent_1$ 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".

Algorithm 4.2 Pseudo code of crossover for evolving 4T1R-based reconfigurable architecture $crossover()$

```
parent_1, parent_2 selection by tournament strategy
if parent_1.size() != parent_2.size() then
  if random < P_c then
    if parent_1.size() > parent_2.size() then
      offspring = parent_1
    else
      offspring = parent_2
    end if
  else
    if parent_1.fitness > parent_2.fitness then
      offspring = parent_1
    else
      offspring = parent_2
    end if
  end if
  for w_ij in offspring do
    if w_ij is in overlapped area then
      w_ij ← wights_update()
    end if
  end if
end if
```

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 (4.10).

In a conclusion, the EHW configuration signals are the values of the matrix Wres in the genome of the proposed memristive reservoir circuit. They represent the pulse width of the control voltage V_c applied to the reconfigurable memristive units (RMUs) in the circuit. By changing the pulse width of V_c, the current flowing through the memristors can be controlled to exhibit different nonlinear and fading dynamics, which are essential for the reservoir computing.

The EHW configuration signals are evolved by a scalable evolutionary algorithm,

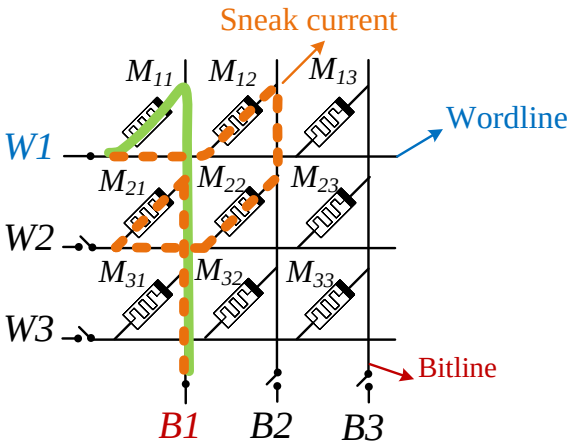


Figure 4.7: Schematic illustration of 1R memristor-based crossbar and sneak current. The green line denotes the desired path of generated current enabled by wordline W_1 and bitline B_1, the orange dot line denotes the sneak current through different undesired paths.

which aims to optimize the memristive reservoir circuit for a given task. The algorithm uses a binary adjacency matrix W_{bool} to represent the topology of the reservoir, and evolves both W_{bool} and W_{res} simultaneously. The algorithm also uses a sparse initialization method based on cycle reservoirs with regular jumps (CRJ) to ensure the circuit feasibility and scalability. The algorithm applies genetic operators such as crossover, mutation, and adaptation to generate diverse and adaptive candidate reservoirs. The fitness of each candidate reservoir is evaluated by simulating the circuit netlist using NGSPICE, and then applying ridge regression to calculate the output weights W_{out} and the actual output y. The fitness is measured by the mean squared error between y and the desired output. For example, suppose the task is to generate a sine wave based on the input signal. The algorithm will initialize a population of candidate reservoirs with different sizes, topologies, and configuration signals. Then, it will simulate each reservoir circuit and calculate its fitness. The best reservoir circuit will be kept as the champion, and the rest will undergo crossover, mutation, and adaptation to generate new reservoir circuits. This process will be repeated for a number of generations until a satisfactory solution is found or a termination criterion is met. The final solution will be a memristive reservoir circuit that can generate a sine wave with high accuracy and low

power consumption.

4.2.2 Evolving 1R-based Reconfigurable Architecture

1R-based Memristive Reconfigurable Architecture

Besides 4T1R memristive architecture, we also apply the 1R memristor-based crossbar to construct the reconfigurable architecture for RC, which is illustrated in Figure 4.7. Unlike the 4T1R architecture, which requires four transistors for each memristor, the 1R architecture is a passive array that connects the memristors directly to the word lines and bit lines. This reduces the area and power consumption of the array, but also introduces some challenges such as sneak paths and nonlinearity. The design of the 1R memristor-based crossbar can be defined as a 3-tuple $\mathbb{C} = (\mathbb{M}, \mathbb{W}_x, \mathbb{W}_y)$, where

- $\mathbb{M} = \begin{pmatrix} m_{11} & m_{12} & \cdots & m_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ m_{l1} & m_{l2} & \cdots & m_{ln} \end{pmatrix}$ is a two-dimensional array of memristors consisting of l rows and n columns, where $m_{ij} \in (0, 1)$ denotes the states of the memristor connecting row l to column j. The horizontal connections are called wordlines, and the vertical connections are called bitlines.

- $\mathbb{W}_X = \{x_1, x_2, ..., x_i, ..., x_l\}$ is the switch vector applied for wordline, where $x_i \in \{0, 1\}$ represents the signal state applied to the i-th wordline (row).

- $\mathbb{W}_y = \{y_1, y_2, ..., y_i, ..., y_n\}$ is the switch vector applied for bitline, where $y_i \in \{0, 1\}$ represents the signal state applied to the j-th bitline (column).

In a 1R memristor-based crossbar array of size $l \times n$, l is the number of rows or wordlines, l_open is the number of open wordlines, and l_closed is the number of closed

wordlines. n denotes the number of columns or bitlines, n_open denotes the number of open bitlines, and n_closed denotes the number of closed bitlines. The switch vector set for a 1R memristor crossbar array represent the connection states of the signals to the row and column. A closed wordline is one that is connected to an input voltage source, whereas an open wordline is one that is not connected to an input voltage source. x_i denotes the i-th row's switch state, where "1" is closed and "0" is open. A bitline closed indicates that the grounded current sensor on that column output is connected to that bitline, whereas a bitline open indicates that no grounded output current sensor is connected to that bitline. A bitline closed is called a selected bitline. y_j is the switch state for the j-th column, where "1" is closed and "0" is open. Therefore, the states of $x_1, x_2, ..., x_i, ..., x_l$ and $y_1, y_2, ..., y_i, ..., y_n$ are combined to define the switch vector to enable the 1R memristor-based crossbar. Besides the switch vectors applied to the memirstor-based crossbar, the state initialization of the memristor m_{ij} is also involved in the memristor-based crossbar design. The total number of sneak currents in a $l \times n$ crossbar will be $(2^l - 2) \times (2^n - 2)$.

We also adopt the memristor model proposed by [45] to construct the nonlinear and fading memory of the memristive reservoir, which has been introduced in Section 4.1.1. Figure 4.8 visualizes the traces of sneak currents existing in a 3×3 1R memristor-based crossbar and its equivalent circuit, where only $W1$ and $B1$ are enabled.

There will be different sneak currents in the memristor-based crossbar from $W1$ to $B1$ based on the difference of voltage potential. These sneak currents enable series connections in the memristors-based crossbar, and its equivalent circuit of memrsitor crossbar is shown in the right part of Figure 4.8. For example, there is a green trace of sneak current flowing across M_{13}, M_{23} and M_{22}, respectively. These sneak currents enable connecting the memristors in a cyclic way, which is consistent with a memristive reservoir with simple cyclic topology such as the one proposed in [287]. Moreover, the currents flowing across M_{12} and M_{21} are negative and the one flowing across M_{21} is positive. Together with the cyclic topology, they

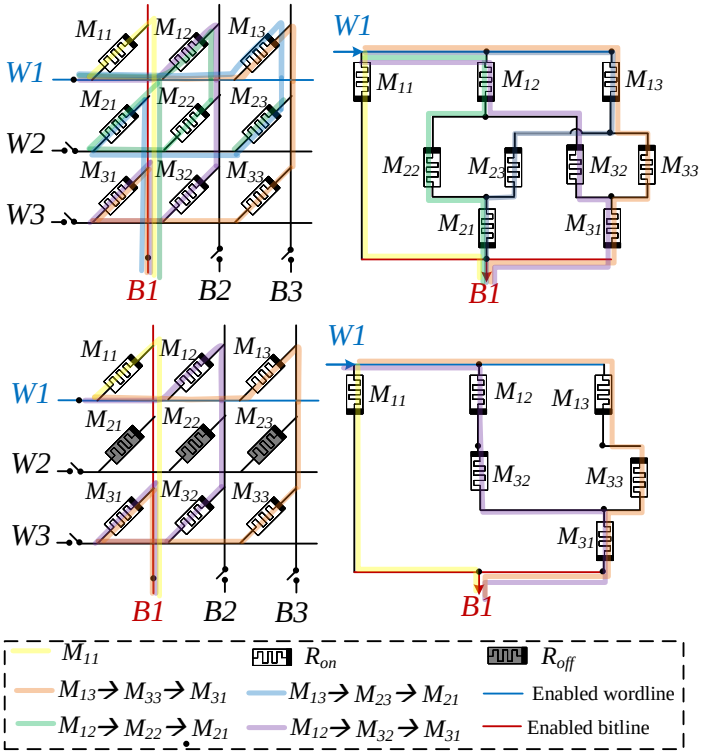


Figure 4.8: The different traces of sneak currents in one 3×3 1R memristor-based crossbar and its equivalent circuit.

can transfer the input signal into a high-dimensional state space [288]. With the increasing size of the memristor-based crossbar, there will be more memristors constructing a larger cycle for reservoir.

Therefore, 1R memristor-based crossbar is capable of meeting the features of memristive reservoir circuit. We will apply the temporal sequence of the current flowing across the memristors in the memristive crossbar as the high-dimensional spatiotemporal patterns generated by the memristive reservoir.

Indirect Circuit Representation for 1R-based Reconfigurable Architecture

As mentioned in the previous Section 4.1, the traditional reservoir computing paradigm consists of an input layer, a reservoir, and an output layer. As for the input layer, we will apply the assignment strategy of input signals as alternatives to the input layer of the

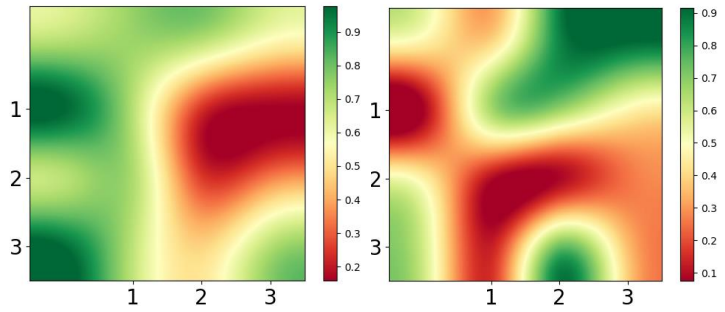


Figure 4.9: The visualization of two different scenarios of memristor states in a 1R memristive crossbar. X axis and Y axis represent the index of bitline and wordline, respectively.

traditional model, where the assignment strategy of input signals is divided into two parts, the input-connection states W_{bool} and the input-injection timing states W_{tim}, respectively. By evolving the assignment strategy of input signals, the raw input signals can be masked adaptively. And as for the output layer of the reservoir, W_{out} is set by offline training through ridge regression using Equation (4.7).

As the memristive reservoir computing model introduced in Section 4.1, the states of reservoir nodes are represented by the states of the current flowing through the memristors. In addition, the currents flowing through the memristors are highly related to the memristor state x according to the Equation (4.1) to (4.2). Therefore, besides evolving the assignment strategy of input signals, the distribution strategy of memristor states, which is represent as W_{mem}, also needs to be evolved.

By evolving the assignment strategy of input signals and the distribution strategy of memristor states, the sneak currents generated in the 1R memristor-based crossbar can be manipulated, which will further enable the memristive crossbar-based reservoir with adaptive mask and modularity. We summarise what is required to be evolved to construct the reservoir circuit in Table 4.3.

Mutation

In order to encourage diversity of manipulating the sneak currents during the evolution, four

Table 4.3: Genome of proposed 1R memristor-based RC

Genes	Meaning	Size*	Variable Type
W_{bool}	Input-connection states	$2N \times 1$	Boolean
W_{tim}	Input-injection timing	$2N \times 1$	Real value
W_{mem}	Memristor states	$N \times N$	Real value

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

types of mutation operators are applied based on the mutation rate P_m, and the algorithm that how to apply these four operations is presented in Algorithm 4.3:

- Input-connection states mutation (`mutate_connection`): The input-connect states are represented by W_{bool}. Each value in the W_{bool} will be flipped with probability P_{con1}. The pseudo code of this mutation operation is presented in Algorithm 4.4.

- input-injection timing mutation (`mutate_timing`): For the values in W_{tim} corresponding to the position where W_{bool} is not zero, then $W_{tim}[i]$ will be mutated with probability probability P_{tim1} to a new value taken uniformly at random within the allowable range ([0,1]). The pseudo code of this mutation operation is presented in Algorithm 4.5.

- Memristor state mutation (`mutate_states`): Memristor states are represented by W_{mem}. The value of $W_{bool}[i][j]$ will be mutated with probability P_{mem1} to a new value taken uniformly at random within the allowable range ([0,1]). The pseudo code of this mutation is presented in Algorithm 4.6.

- Increase size (`mutate_size`): This operator increases the size of the crossbar by adding one row and one column to the crossbar. The new memristor states $W_{mem}[i][j]$ corresponding to the new row and column are initialized to random values within the allowable range, and its corresponding input-connection states are initialized to 0. The

Algorithm 4.3 Pseudo code of mutation for evolving 1R-based reconfigurable architecture `mutate()`

Input: *genomes* in the current generation, P_m, P_{con}, P_{tim}, P_{sta}, P_{siz}, P_{con1}, P_{tim1}, P_{sta1}, P_{siz1}

Output: *genomes* after mutation operations

```
1: if random() < P_m then
2:   if random() < P_con then
3:     genome ← mutate_connection(genome, P_con1)
4:   end if
5:   if random() < P_tim then
6:     genome ← mutate_timing(genome, P_tim1)
7:   end if
8:   if random() < P_sta then
9:     genome ← mutate_states(genome, P_sta1)
10:  end if
11:  if random() < P_siz then
12:    genome ← mutate_size(genome, P_siz1)
13:  end if
14: end if
15: Return genome
```

pseudo code of this mutation operation is presented in Algorithm 4.7.

Regarding the diversity of sneak currents, our four proposed mutation operations can mutate the crossbar size, memristor states, input-connection states, and timing, which is advantageous for the diversity of memristive reservoir circuits.

Crossover

Algorithm 4.8 displays the pseudocode of the crossover operation. The crossover probability is P_c, which is to decide whether to apply the crossover operation (Line 2). The same pseudocode is used both for the crossover of the W_{mem} component of the genome, and for the crossover of the W_{tim} and W_{bool} components of the genome. The crossover of W_{tim} and W_{bool} is performed jointly at the same time. This is achieved by considering that a parent corresponds to the product of each element of W_{tim} by each corresponding element

Algorithm 4.4 Pseudo code of input-connection state mutation for evolving 1R-based reconfigurable architecture `mutate_connection()`

Input: *genomes* that will be taken the `mutate_connection()`, P_{con1}
Output: *genomes* after `mutate_connection()`

```
for i in 2N do
    if random() < P_con1 then
        W_bool[i] ← 1 − W_bool[i]
    end if
end for
Return genomes with mutated W_bool
```

Algorithm 4.5 Pseudo code of input-injection timing mutation for evolving 1R-based reconfigurable architecture `mutate_timing()`

Input: *genomes* that will be taken the `mutate_timing()`, P_{tim1}
Output: *genomes* after `mutate_timing ()`

```
for i in 2*N do
    if W_bool[i]! = 0 then
        if random() < P_tim1 then
            W_tim[i] ← random.uniform(0, 1)
        end if
    end if
end for
Return genomes with mutated W_tim
```

Algorithm 4.6 Pseudo code of memristor state mutation for evolving 1R-based reconfigurable architecture `mutate_states()`

Input: *genomes* that will be taken the `mutate_states()`, P_{mem1}
Output: *genomes* after `mutate_states()`

```
for i in N do
    for j in N do
        if random() < P_mem1 then
            W_mem[i][j] ← random.uniform(0, 1)
        end if
    end for
end for
Return genomes with mutated W_mem
```

of W_{wool} of a given individual.

For the crossover of parents with different sizes, the size of the offspring's reservoir should be determined firstly. There are two alternatives of determining the size of offspring's

Algorithm 4.7 Pseudo code of increase size mutation for evolving 1R-based reconfigurable architecture `mutate_size()`

Input: *genomes* that will be taken the `mutate_size()`, P_{siz1}
Output: *genomes* after `mutate_size ()`

1: Initializing $temp\_row\_mem$ in $[0, 1]$ with size $1 \times N$. Initializing $temp\_con\_mem$ in $[0, 1]$ with size $(N + 1) \times 1$
2: Stacking W_{mem} with $temp\_row\_mem$
3: Stacking W_{mem} with $temp\_con\_mem$;
4: Initializing $temp\_bool$ with 0
5: Stacking W_{bool} with $temp\_bool$;
6: N+=1
7: **Return** *genomes* with mutated W_{mem} and W_{bool}

reservoir, which are to follow the parent with larger size and to choose the size of the parent with better fitness, respectively. This is shown in Lines 3-15.

The weights in the overlapping area are averaged or selected from either parent also based on Equation (4.10) (Line 18 of Algorithm 4.8), where w^1 represents the weights from parent 1 and w^2 represents the weights from parent 2. Based on the size of the reservoir, weights in the excess area are either completely adopted or completely discarded. This is shown in Line 20 of Algorithm 4.8.

4.2.3 Proposed Evolutionary Algorithm

The overall flowchart of the evolutionary memristive reservoir circuit is shown in Figure 4.10. First, the population of candidate reservoirs is initialised with num_Pop individuals representing different circuits. After that, the circuit netlists in SPICE language corresponding to the individuals are generated automatically. These netlists will be used by the circuit simulator NGSPICE to employ and execute the circuit simulations (dark grey box in the figure). Then the results of the circuit simulation are used for fitness evaluation, which indicates how well the circuit implements the desired task. Following that, two parent individuals are selected to undergo crossover and mutation operations specifically designed

Algorithm 4.8 Pseudo code of crossover for evolving 1R-based reconfigurable architecture `crossover()`

Input: *genomes* in the current generation, P_c
Output: *genomes* after crossover operations

1: $parent_1, parent_2$ selection by tournament strategy
2: **if** $random() < P_c$ **then**
3: **if** $parent_1.size() != parent_2.size()$ **then**
4: **if** $parent_1.size() > parent_2.size()$ **then**
5: $temp\_off \leftarrow parent_1$
6: **else**
7: $temp\_off \leftarrow parent_2$
8: **end if**
9: **else**
10: **if** $parent_1.fitness > parent_2.fitness$ **then**
11: $temp\_off \leftarrow parent_1$
12: **else**
13: $temp\_off \leftarrow parent_2$
14: **end if**
15: **end if**
16: **for** each element w_{ij} in $temp\_off.size()$ **do**
17: **if** w_{ij} is in overlapped area **then**
18: Set w_{ij} based on Equation (4.10), where w_{ij}^1 and w_{ij}^2 are the corresponding elements from $parent_1$ and $parent_2$, respectively
19: **else**
20: Discard w_{ij}
21: **end if**
22: **end for**
23: **Return** $temp\_off$
24: **end if**
25: **Return** $parent_1$

for evolving memristive circuits. These two parents are selected by tournament selection with tournament size of num_Tour. The crossover and mutation operations will generate one new child individual. The best num_Pop individuals according to their fitness survive for the next generation. This procedure is repeated for a maximum number of generations num_Gen. More details about the population initialization, circuit netlist generation, and evolution algorithm design will be given in the following subsections.

The evolutionary designs of memristive reconfigurable architectures for RC are based on two types of memristive arrays, which are 4T1R memristor-based crossbar and 1R memristive-based crossbar, respectively. They are all encoded by the graph representa-

Algorithm 4.9 Pseudo code of search for crossbar design for reservoir computing SEARCH_XBAR()

Input: population_Num: N, maxium_Iteration: max_Iter, fitnessFunction: f, probability of crossover P_c, probability of mutation P_m

Output: M, Φ

```
Initialise population p
for each generation g do
    for each genome in p do
        circuit ← phenotype(genome)
        fitness_genome ← EVAL_XBAR(circuit_netlist)
    end for
    champion ← clone of genome with best fitness
    p[0] ← champion
    for genome in p[1, N) do
        parent_1, parent_2 ← selection(all genomes)
        genome ← crossover(parent1, parent2, P_c)
        genome ← mutate(genome, P_m)
    end for
end for
Return M, Φ
```

tion, therefore, we adopt the same evolutionary algorithm loop to evolve these two types of memristive arrays, which is shown in the Algorithm 4.9. Algorithm 4.9 describes the pseudo code of our proposed reservoir circuit evolution, which gives a procedure that how to search an optimal circuit configuration or assignment strategy for the memristive crossbar, SEARCH_XBAR. The first step is to populate the system. All candidate reservoirs will be generated as introduced in Section 4.2.1 (4T1R-based reservoir) and Section 4.2.2 (1R-based reservoir) , so that the circuit netlists for training and test will be further generated. Specifically, as for the 4T1R-based reservoir, all the individuals will be initialized by the sparse initialization as introduced in Section 4.2.1. As for the 1R-based reservoir, the initial size of 1R memristor-based crossbar is predefined, its corresponding memristor states and the input-injection timing are initialized uniformly at random in the range of $(0, 1)$, and the input-connection states are initialized randomly either 0 or 1. Then their fitness will be evaluated by the function EVAL_XBAR(), which is depicted in Algorithm 4.10. Their performance will be recorded as fitness, and the champion gene will be preserved. The fitness

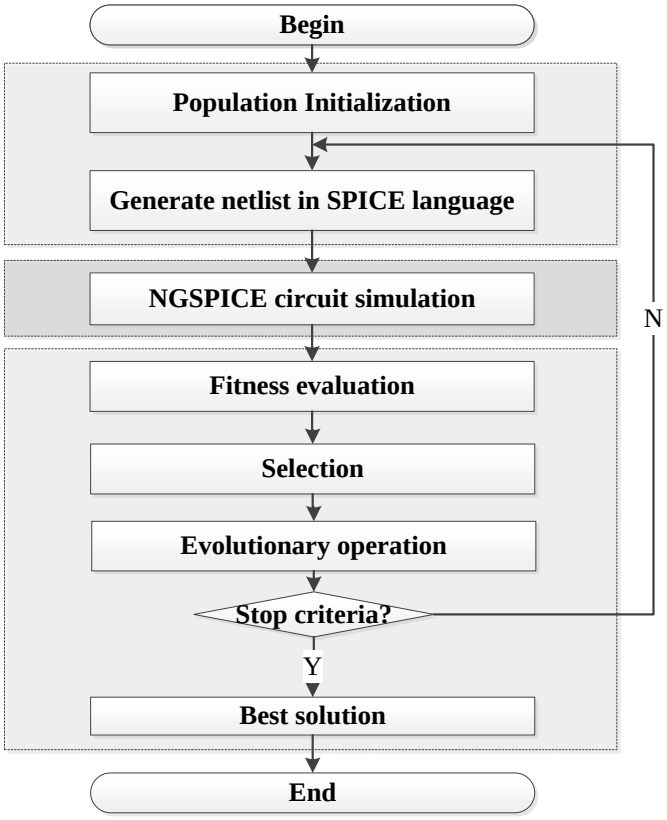


Figure 4.10: Overall flowchart of circuit evolution.

equation is as follows:

$$fitness = -100 + A\sqrt{\left\langle \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\|^2 \right\rangle}, \tag{4.11}$$

where A is the scaling factor, which is selected according to the magnitude of the task to be performed. The value $\mathbf{y}(t)$ is the desired output (target), and $\hat{\mathbf{y}}(t)$ is the readout output.

After the fitness evaluation (Line 2 to Line 6 of Algorithm 4.9), the best gene will be kept (Line 7 of Algorithm 4.9). Next, two parents are selected and they go through mutation and crossover with probability P_c and P_m, respectively. Their details have been introduced in Sections 4.2.1 and 4.2.2.

Algorithm 4.10 Pseudo code function for evolving 1R-based reconfigurable architecture `EVAL_XBAR()`

Input: circuit_netlist, **y**
Output: f

1: $res\_out \leftarrow$ `NGSPICE`$(circuit\_netlist)$
2: **if** $res\_out \neq 0$ **then**
3: Calculate the weight of output layer W_{out} according to Equation (4.7)
4: $\hat{\mathbf{y}} \leftarrow res\_out \times W_{out}$
5: $f \leftarrow -100 + A\sqrt{\langle \|\hat{\mathbf{y}} - \mathbf{y}\|^2 \rangle}$
6: **end if**
7: **Return** f

4.3 Experimental Studies of 4T1R-based Reconfigurable Architecture for RC

This section will introduce the experimental studies of 4T1R-based evolutionary design for RC. In this section, we will verify our proposed evolvable 4T1R memristive reservoir circuit, where the predictive performance and circuit performance will be demonstrated and analyzed. Section 4.3.1 gives the experimental setup, which includes the experimental tasks, circuit setting, and algorithm setting, respectively. Section 4.3.2 presents the ablation studies of evolutionary operations. Section 4.3.3 gives the comparison results with other reservoir models on prediction performance. Section 4.3.4 gives the comparison results with other hardware reservoirs in terms of the circuit area and power consumption.

4.3.1 Experimental Setup

Experimental Tasks

In order to verify the feasibility of our proposed 4T1R-based Evolutionary Design for RC, seven tasks are executed, including one wave generation task and six prediction tasks.

- Wave Generation Task: Memristors have recently been used as single devices or in the form of networks for wave generation tasks [266]. In addition, wave generation tasks also has been commonly applied to verify the feasibility of hardware reservoir computing [287]. The wave generation task used in our experiments is illustrated in Figure 4.11 (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 a 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 Figure 4.11 (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.

- Nonlinear Dynamic System Prediction Task: We considered the NARMA systems of order 10 to verify our proposed method [223]:

$$\begin{aligned} y\left(t+1\right) =& 0.3y\left(t\right) + 0.05y\left(t\right)\sum_{i=0}^{9} y\left(t-i\right) \\ &+ 1.5s\left(t-9\right)s\left(t\right) + 0.1, \end{aligned} \tag{4.12}$$

  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.

- 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 [90]. This dataset is from [115], 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.

- ARFIMA Series Prediction Task: We generated the ARFIMA series using the following model with long memory effect [89]:

$$(1 - 0.7B + 0.4B^2)(1 - B)^{0.4}Y_t = (1 - 0.2B)\varepsilon_t, \tag{4.13}$$

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.

- 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 [2], 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.

- Dow Jones Industrial Average (DJI) Prediction Task: The raw dataset contains DJI daily closing prices from 2000 to 2019 obtained from Yahoo Finance [3], 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.

- Santa Fe Laser Prediction Task: Santa Fe Laser data set was used [4], which consists of a cross-cut through periodic to chaotic intensity pulsations of a real laser. The length

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

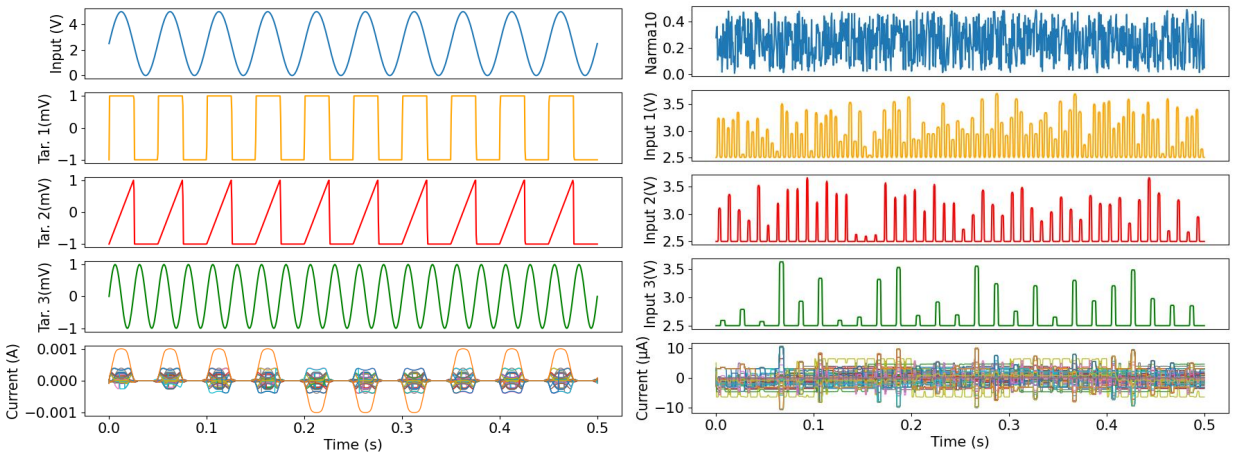


Figure 4.11: Example visualization of related signals of memristive reservoir circuit. (a) Wave generation task: Input sine wave, three target waves and current signals of memristive reservoir circuits; (b) Narma-10 system prediction task: Target series, three discretized input signal and current signals of memristive reservoir circuits.

of training and test set are both 2000, where the first 200 values from them are used as the initial washout period.

- Dynamic Gesture Recognition (DGR): This consists of spatioatemporal time series data for the task of recognizing dynamic gestures [358]. These dynamic gestures are recorded in three dimensions at a sample frequency of 10 Hz. Typical 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.

Algorithm Setting

In our experiments, Root Mean Squared Error (RMSE) is used as a measure of predictive performance [9]:

$$RMSE = \sqrt{\langle \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\|^2 \rangle}, \tag{4.14}$$

where $\mathbf{y}(t)$ is the desired output (target), $\hat{\mathbf{y}}(t)$ is the readout output, $\|.\|$ denotes the Eu-

clidean norm, and $\langle . \rangle$ denotes the empirical mean. Besides RMSE used in our work, average rank is also applied to compare with other SoTA works, where average rank is a relative measure derived from the RMSE values. It is calculated by first ranking each model according to its RMSE value across all test cases, with the model demonstrating the lowest RMSE receiving the highest rank.

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 [85], ESN [128], vanilla Long Short Term Memory (LSTM) [111], memory-augmented LSTM and memory-augmented RNN [354]. These approaches have been chosen as baseline for time series information processing widely [354] [233].

We have applied not only grid search, but also differential evolution, which is a classic optimization method for searching optimal parameters [316, 283], 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 4.1 *Adapt()*, and is set as 0.3. The parameter A used in Equation (4.11) 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 [312].

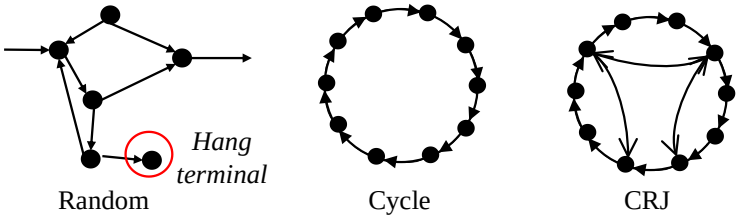


Figure 4.12: Different topologies of reservoir.

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.

Circuit Setting

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 Figure 4.12. 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

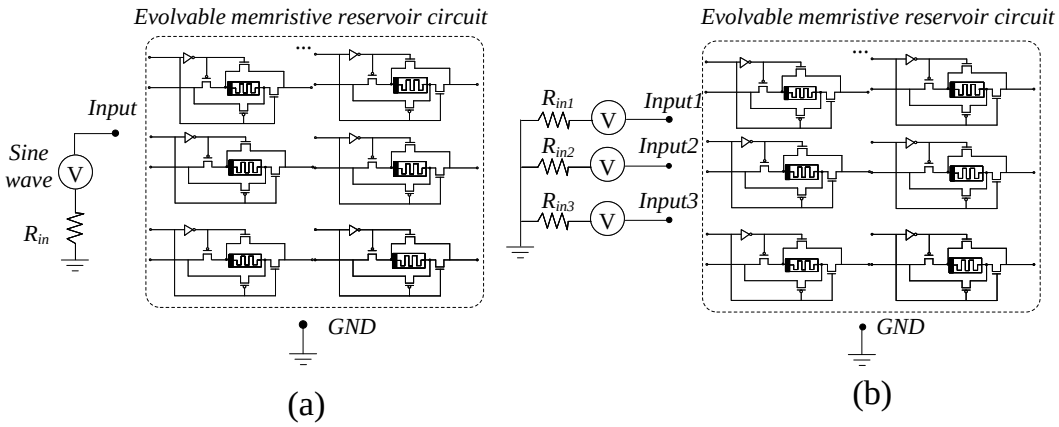


Figure 4.13: The diagrams of embryo circuits for evolving memristive reservoir circuit to wave generation (a) 10th-order Narma dynamic system prediction (b).

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 Figure 4.13 (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 $s\,(k)$ before feeding it into the memristive reservoir circuit. The amplitude of the input signal $s\,(k)$ is linearly converted into a voltage pulse with amplitude $V_{in}(k)$ that is then applied to the memristor reservoir:

$$V_{in}(k) = 2.5 \times s(k) + 2.5. \tag{4.15}$$

This linear conversion allows the input voltage pulses to fall in the range of $2.5\breve{}5V$

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 [66] [67]. 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 Figure 4.13 (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, Figure 4.11 (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.

4.3.2 Ablation Studies of Evolutionary Operations

The Effect of *adapt* Operation

In this section, we investigate the effectiveness of different operations in our proposed algorithm. Table 4.4 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 RMSE 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 4.5.

From Table 4.4, 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.

The Effect of *mutation* and *crossover* Operation

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 4.4, after removing the crossover or mutations, the performance is worse than those of the 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 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 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 4.4. We can see that removing each individual mutation operation leads to performance degradation. According to the results shown in Table 4.4, 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 in the circuit evolution. Other mutation operations also have an impact on the circuit evolution.

Then, we investigate the impact of memristor variability on the performance. Table 4.4 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 4.4, 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.

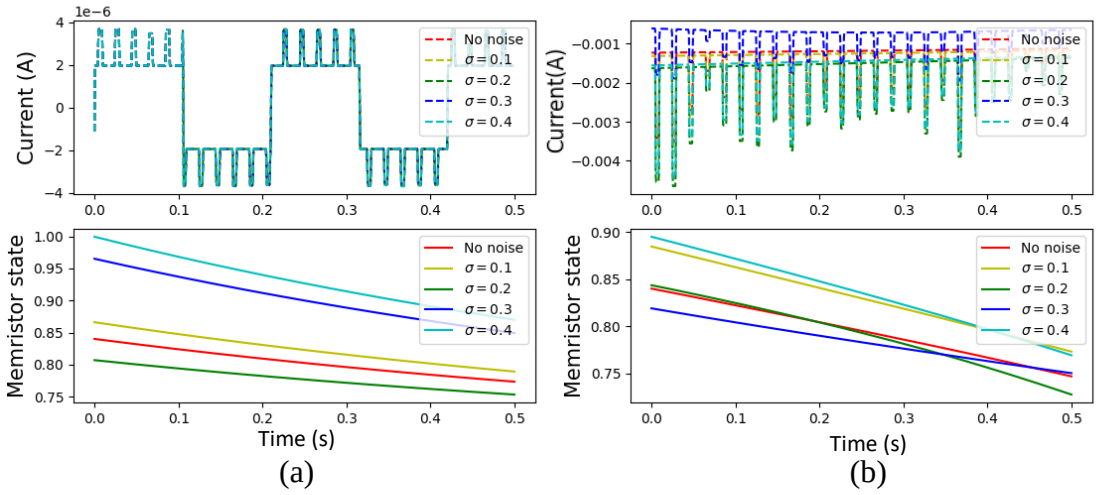


Figure 4.14: (a) The visualization of the current and memristior state of one memristive element based on our proposed reconfigurable memristive reservoir. (b) The visualization of the current and memristior state of one memristive element based on pure memristive reservoir.

Moreover, we also investigate the noise impact on the circuit perspective of memristive reservoir. Figure 4.14 shows the current and memristor states of one memristive element based on our proposed reconfigurable memristive reservoir (Figure 4.14 (a)) and the pure memristive reservoir (Figure 4.14 (b)) under noise injection with different σ. As shown in Figure 4.14, 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 [156], 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.

Table 4.4: 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	-

4.3.3 Comparisons With Other Reservoir 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 4.5.

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 better performance than the cycle memristive reservoir. And the memristive reservoir evolved by our proposed approach outperforms both of the cycle and CRJ memristive reservoir circuits.

Moreover, in order to further verify the performance of our proposed memristive

Table 4.5: Comparisons of memristive reservoir circuits with different topologies

	Implemen.	Cir. feas.	Opti. Method	Evo.?	Wave.* (#Node=60)	DGR	Nar. 10	DJI	Audio	Tree	ARF	Santa	Ave. Ran*.
Baseline approaches of RC													
RNN [85]-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 [128]-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 [111]-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 [354]-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 [354]-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 [85]-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[128]-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[111]-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 [354]-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 [354]-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.

reservoir circuits, several baseline approaches of RC are compared as reference, which are vanilla RNN [85], ESN [128], vanilla LSTM [111], memory-augmented LSTM and memory-augmented RNN [354]. As for the latter two, another parameter D is introduced for the memory augmentation, therefore, memory-augmented LSTM 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 4.5 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 4.5, 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 4.5. 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 Figure 4.16. 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 4.2.1 (an example of Jazz task is shown in Figure 4.15), 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

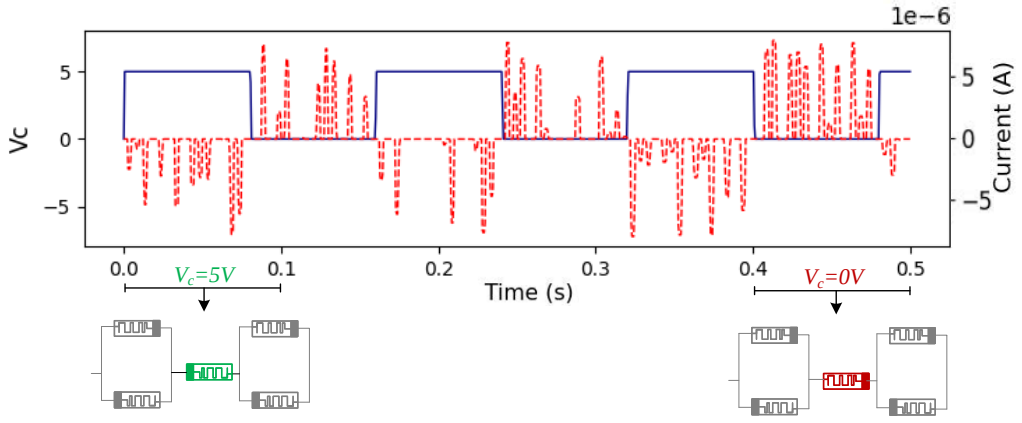


Figure 4.15: The current and control signal visualization of one RMU under the task of audio, and its equivalent circuit state under different V_C states.

states. Taking Figure 4.16 (a) as an example, 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 [354]. And, the connections with short dependency memory (lighter color cubes) tend to be generated to the reservoir circuits for solving the tasks with short-term memory, such as Narma 10.

We can also see from Figure 4.16 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.

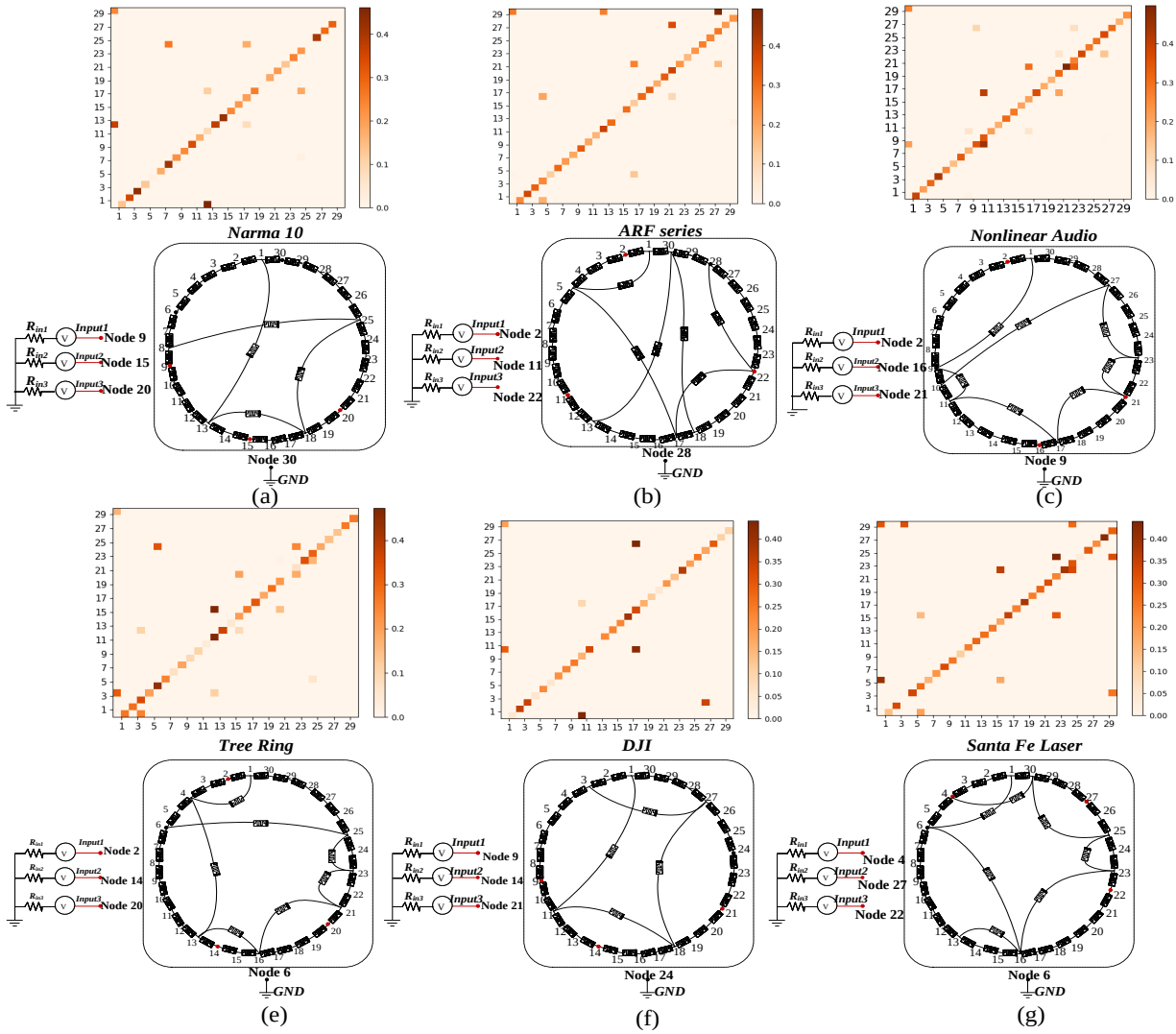


Figure 4.16: Visualization of memristive reservoir topology and equivalent circuits. (a) Narma-10; (b) Nonlinear audio; (c) ARFIMA series; (d) Tree ring; (e) DJI (f) Santa Fe laser.

Figure 4.17 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.

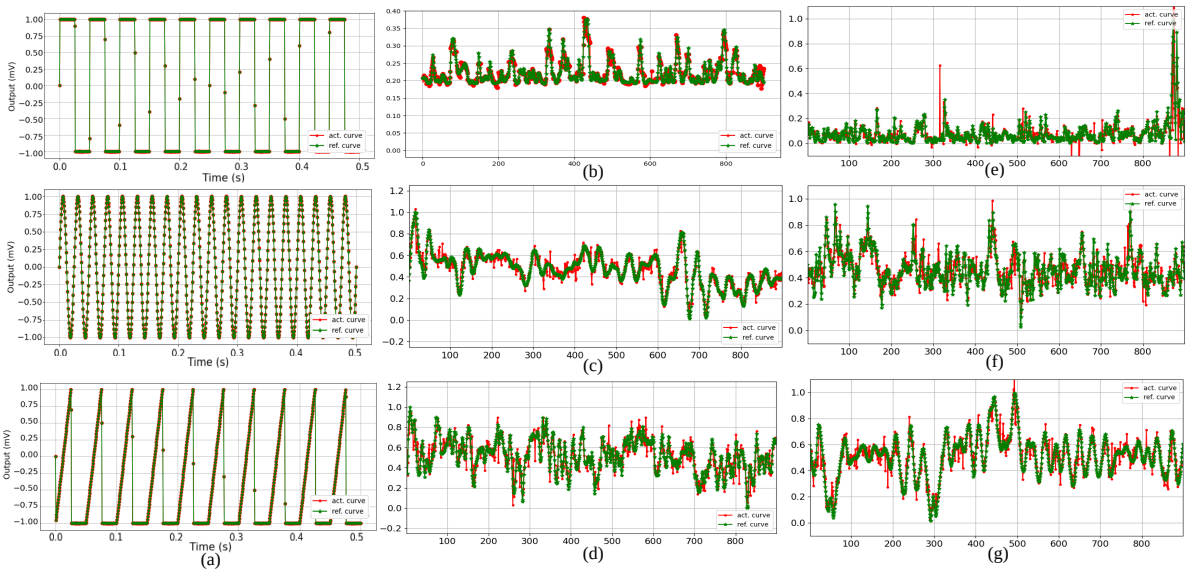


Figure 4.17: Actual outputs of our proposed memristive reservoir vs corresponding targets. (a) Wave generation task; (b) Narma-10; (c) Nonlinear audio; (d) ARFIMA series; (e) Tree ring; (f) DJI (g) Santa Fe laser.

Table 4.6: 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?
[286]	Manually design	Sepcified memristive circuit	No	No	No
[156]	Manually design	Sepcified memristive circuit	No	No	No
[171]	Manually design	Sepcified atomic switch circuit	No	No	No
Our work	**Evolutionary approach**	**Reconfigurable memristive circuit**	**Yes**	**Yes**	**Yes**

4.3.4 Evolved Circuit Performance Comparisons

Table 4.6 lists the comparison of existing methods with our proposed method. We have compared our proposed work with the existing methods [286] and [171], [156] by multiple perspectives, which are design method, hardware characteristics, and reservoir characteristics.

Regarding the design method, the reservoir of other existing work [286] [156] [171] 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.

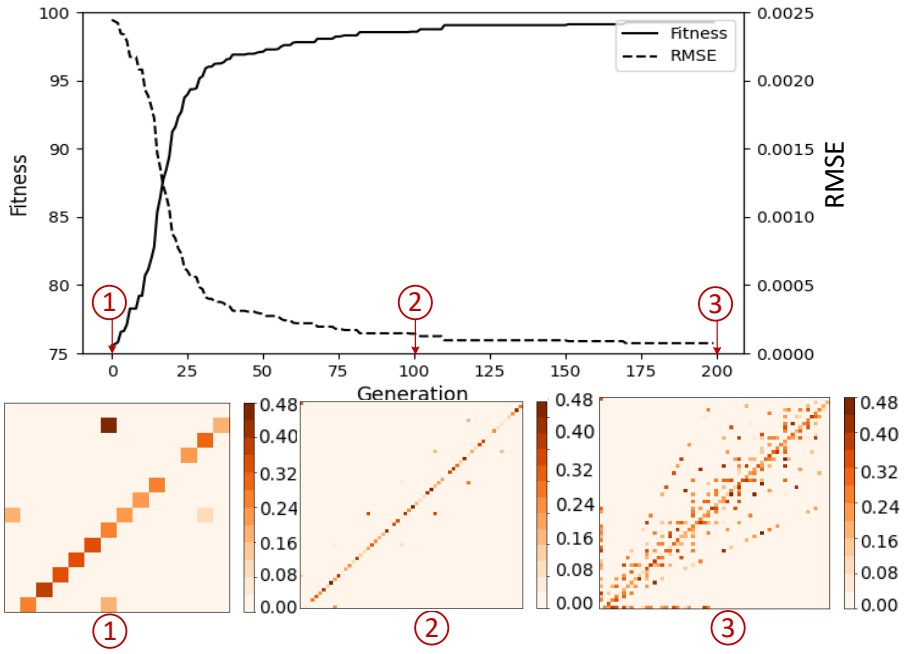


Figure 4.18: Visualization of the network structure changes with the generation and its corresponding fitness results

Table 4.7: Comparison of different hardware reservoirs in applied components

Methods	Implementation	Components used to construct a reservoir		
		#Memristor*	#MosFets	#Capacitor
FPGA-based ESN [344]	FPGA	0	37n*	n
Mosfet crossbar-based ESN [158]	PCB	0	16n	0
CMOS-based LSM [238]	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.

Regarding the hardware characteristics, other works [286, 171, 156] 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 s earching for the optimal reservoir architecture for different tasks, leading to better predictive performance. In contrast, the existing methods [286, 171, 156] are based on a fixed circuit architecture that cannot be reconfig-

ured, which can limit their predictive performance. Moreover, for other work [286, 171, 156], 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. Figure 4.15 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 4.7 shows the comparisons of different hardware reservoirs. We can see that ESN or LSM models have to use n 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 [344] 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 [344, 158, 238], by manipulating the currents flowing across the memristiors, 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, Figure 4.18 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 Figure 4.18, 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 Figure 4.18 (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.

4.4 Experimental Studies of 1R-based Reconfigurable Architecture for RC

This section will introduce the experimental studies of 1R-based evolutionary design for RC. We will verify the our proposed evolvable 1R memristive reservoir circuit in terms of its predcitve performance and circuit performance. Section 4.4.1 gives the experimental setup, which includes the experimental tasks, circuit setting, and algorithm setting, respectively.

Section 4.4.3 presents the ablation studies, which include the effect of the adaptive mask and the effect of the modularity, respectively. Section 4.4.4 gives the comparison results with other reservoir models on prediction performance. Section 4.4.5 gives the comparison results of resource consumption.

4.4.1 Experimental Setup

Experimental Tasks

Both single-task problems and the multitasking problems are used to verify our proposed method. Each single-task problem (Narma 10, Narma 20, Jazz and Tree) is described below. The multitasking problems are defined based on the single tasks as follows: two tasks involve predicting Narma 10 and Narma 20 concurrently, three tasks involve predicting Narma 10, Narma 20 and Jazz concurrently, and three tasks involve predicting Narma 10, Narma 20 and Tree concurrently.

- Narma 10 – A 10-order Nonlinear Dynamic System Prediction Task: The Nonlinear Auto-Regressive Moving Average (NARMA) system is a discrete time system. This system was first implemented in [10]. The current output is determined by the input as well as the previous output. Because of the nonlinearity and possibly long memory, modeling this system is difficult in general. We considered the NARMA systems of order 10 to evaluate our proposed method [10]:

$$\begin{aligned} y\left(t+1\right) =& 0.3y\left(t\right) + 0.05y\left(t\right)\sum_{i=0}^{9} y\left(t-i\right) \\ &+ 1.5s\left(t-9\right)s\left(t\right) + 0.1, \end{aligned} \tag{4.16}$$

  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 systems. We used a NARMA sequence with 2000 items. The first 1000 were used for training, and the remaining were used for testing. The first 200 values from the training and test sequences were used as the initial washout period.

- Narma 20 – A 20-order Nonlinear Dynamic System Prediction Task:

  We also consider the NARMA systems of order 20 to evaluate our proposed method [10]:

$$\begin{aligned} y\left(t+1\right) = \tanh\left(0.3y\left(t\right) + 0.05y\left(t\right)\sum_{i=0}^{19} y\left(t-i\right)\right. \\ \left. + 1.5s\left(t-19\right)s\left(t\right) + 0.1\right), \end{aligned} \tag{4.17}$$

  We also used a NARMA sequence with 2000 items, where the first 1000 were used for training, and the remaining were used for testing. The first 200 values from the training and test sequences were used as the initial washout period.

- Jazz – A 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, when a sequence of consecutive samples is missing, or when impulsive noise appears. Researchers found that long memory appears to be strongly represented in music [90]. This dataset is from [115], 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 from the training and test sequences were used as the initial washout period.

- Tree – A Tree Ring Prediction Task: This dataset contains 4351 tree ring measures of a pine tree from an Indian Garden, Nevada Gt Basin, obtained from tsdl [5], where 2500 items are used for training, and 1850 for testing. The first 200 values from them were used as the initial washout period.

[5] https://pkg.yangzhuoranyang.com/tsdl/

Table 4.8: Parameter setting of baseline approaches and our proposed work

Models	Parameter setting
RNN	hidden size=64 or 100; nonlinearity='tanh'
ESN	hidden size=64 or 100; nonlinearity='tanh'; leaking_rate=1,spectral_radius=0.9
Our work	num_Gen=200; num_Pop=40; num_Tour=3; P_m=0.8; P_c=0.5; num_ini_node=7; num_max_node=10; A=10000

Algorithm and Circuit Setting

With the objective of evaluating the predictive performance of the proposed approach, we conduct comparisons with two existing approaches for time series prediction, namely ESN [128] and RNN [85]. These two approaches have been chosen as baselines for time series information processing widely. Besides the software approaches, we also conduct comparisons of feature and RMSE with other reservoir circuits with different topology. 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.

The parameter settings of RNN, ESN, and our approach are given in Table 4.8. As for ESN and RNN, the hidden size is set as either 64 or 100 to investigate the effect of different reservoir sizes, and the nonlinearity is set as *tanh*. The leaking rate is set as 1 and the spectral radius is set as 0.9. These values have been chosen by grid search. As for the circuit simulation part, the performance of the circuits is evaluated by simulation using NGSPICE , which is based on Spice3.

4.4.2 Preliminary Experiments of Selecting Memristor Model

As mentioned in Section 4.1.1, Chen's memristor model will be applied to evolve the memristive reservoir computing circuits. However, HP memristor model is also the one that have

Table 4.9: RMSE results for experiments with or without adaptive mask under the noise-free and noise injection scenarios.

Method	Narma-10	Narma-20	Jazz	Tree	Ave. Ran.*
Without mask					
Noise-free	0.1650	0.4106	0.2813	0.2786	4
Noise injection	0.2915	0.5134	0.5361	0.4935	6
With random mask					
Noise-free	0.1972	0.3523	0.2756	0.2493	3
Noise injection	0.2123	0.4211	0.3813	0.3872	5
With adaptive mask					
Noise-free	0.0449	0.0886	0.0959	0.1271	1
Noise injection	0.0845	0.1106	0.1539	0.2099	2
P-value	Adaptive VS Without: 0.00068; Adaptive VS Random: 0.00067				

* Noise injection is applied during the training and test phases. The results obtained under the noise injection are the average result over 10 runs.
* The standard deviation σ of noise is set as 0.2.

been widely applied to construct the memristive reservoir computing circuits [287, 286]. Therefore, in this section, we will apply HP memristor model to construct the memristive reservoir computing circuit as the preliminary experiments of selecting memristor model for evolving memristive reservoir computing. We have added the preliminary simulations of HP memristor model as follows:

- Step 1: The range of the input signal has been limited to ($2.5V$, $5V$). Therefore, we first apply the pulse with the minimum voltage $2.5V$as input to the embryo circuit.
- Step 2: Check if the nonlinearity of the memristor current can be observed in the given time window, which is $0.5s$.
- Step 3: For each parameter related to the nonlinear variation (μ_v, p, D, a), repeat steps 3.1 and 3.2:
    1. Step 3.1: If no nonlinear variation is observed, this means that the parameter is set too large, requiring more simulation time than $0.5s$. Therefore, the parameter is tuned by decreasing it by 1% of its original value.

2. Step 3.2: If the nonlinear variation is observed sharply, this means that the parameter is set too small, requiring less simulation time than $0.5s$. Therefore, the parameter is tuned by increasing it by 1% of its original value.

- Step 4: Based on these tuned parameters, we then apply the pulse with maximum voltage 5V as input to the embryo circuit. Steps 2 and 3 are repeated to further tune these parameters based on the 5V voltage. We also check if the parameters are still suitable for the 2.5V voltage by applying the pulse with minimum voltage 2.5V as input to the embryo circuit. If the parameters are not suitable for both voltages, we adjust them by using a weighted average of the values obtained from the 2.5V and 5V tuning.

- Step 5: R_{on} and R_{off} are related to the rate of state switching. Once suitable values for (μ_v,p,D,a) are chosen, similar steps as Step 3 and 4 will be applied to tune R_{on} and R_{off}. We also check if the values of R_{on} and R_{off} are suitable for both voltages by applying the pulses with minimum and maximum voltages as inputs to the embryo circuit. If the values are not suitable for both voltages, we adjust them by using a weighted average of the values obtained from the 2.5V and 5V tuning.

We also conducted a preliminary experiment that used the HP memristor model to evolve the memristive reservoir computing circuits, and compared it with the one that used the Chen's memristor model [45]. The evolution range of x_{ini} of the HP memristor model was $[0, 1]$, while the other parameters related to the evolution algorithm were the same as in the previous experiment. The predictive performance of the HP memristor model was inferior to that of Chen's memristor model. This might be because Chen's memristor model had a higher fading memory effect, which provided more diverse dynamic behaviors for the reservoir states, and thus enhanced the final performance of the evolved memristive reservoir computing circuits. Therefore, we used Chen's memristor model [45] in the thesis. In other

words, the memristor model [45] was chosen because it is not only equipped with short-term memory (forgetting effect), which is in accordance with the key feature of the reservoir, namely nonlinear dynamic behavior in a short-term memory manner.

4.4.3 Ablation Studies of Adaptive Mask and Modularity

In this section, we will investigate effect of the adaptive mask and modularity to the predictive performance by the ablation study.

The Effect of Adaptive Mask

We verify the effect of adaptive mask from two perspectives, which are to apply different kinds of masks and apply in different noise scenarios, respectively. Table 4.9 show the results that without mask, with random mask, and with evolved mask in the noise-free and noise injection scenarios. The noise effect is taken into consideration on writing the memristor initial states due to the device-to-device variation and the write-to-write variation of the memristor device [94]. In this experiment, the noise followed the Gaussian distribution. The results are based on 10 runs.

We first analyze the results in the noise-free scenario. As we can see, the results with the adaptive mask outperform the one without mask and with random mask, which indicates that the adaptive mask is capable of improving predictive performance in the noise-free scenario.

In terms of the noise scenario, the results shown in Table 4.9 show that the Gaussian noise from the write operation of the memristor states degrades the predictive performance in all cases. However, the adaptive mask evolved by our approach can better alleviate the performance degradation caused by noise injection. In particular, the adaptive mask

Table 4.10: RMSE results of experiments with or without evolving memristor states for single task and multi-tasking prediction.

Method		Narma-10	Narma-20	Jazz	Tree
Single tasks					
Without evolving memristor state		0.1386	0.0109	0.1072	0.1398
With evolving memristor state		0.0449	0.0886	0.0959	0.1271
Multitasking tasks					
Without evolving memristor state	Narma-10+Narma-20	0.1632	0.1150	-	-
	Narma-10+Narma-20+Tree	0.1849	0.1587	-	0.2451
	Narma-10+Narma-20+Jazz	0.2029	0.1934	0.3587	-
With evolving memristor state	Narma-10+Narma-20	0.0375	0.0847	-	-
	Narma-10+Narma-20+Tree	0.0239	0.0873	0.0734	-
	Narma-10+Narma-20+Jazz	0.0263	0.0842	-	0.1197
P-value	With evolving VS Without evolving: 0.00042; Single VS Multi.: 0.0051				

improves the predictive performance more under the noise injection scenario compared with that under the noise-free scenario.

We use the Narma-10 task as an example to explain why the adaptive mask alleviates the performance degradation under the noise injection scenario. Figure 4.19 shows the adaptive masked input signals under Namra 20 prediction task (a) and multi-tasking prediction task (b). According to Figure 4.19 (a), we can see that the adaptive mask can filter some of the input data making it change into zero and can provide some of the input data with different levels of importance to the reservoir. These masked input signals can not only process the data with different importance, but also can incur less impact of noise on the memristive crossbar-based reservoir since some of the useless data has been filtered. Therefore, the adaptive mask will enable a more noise-tolerant memristive reservoir. Figure 4.19 (b) shows the input signals with adaptive masking for multi-tasking prediction tasks (Narma-10, Narma-20 and Tree). There are three tasks input into the reservoir synchronously, where three types of the voltages are applied for injecting the sub-task data, the pulse, the tooth saw, and the sine wave respectively. Similar to the single prediction task, the adaptive mask for the multi-tasking prediction can also filter some the useless data, and can process the input data with different levels of importance.

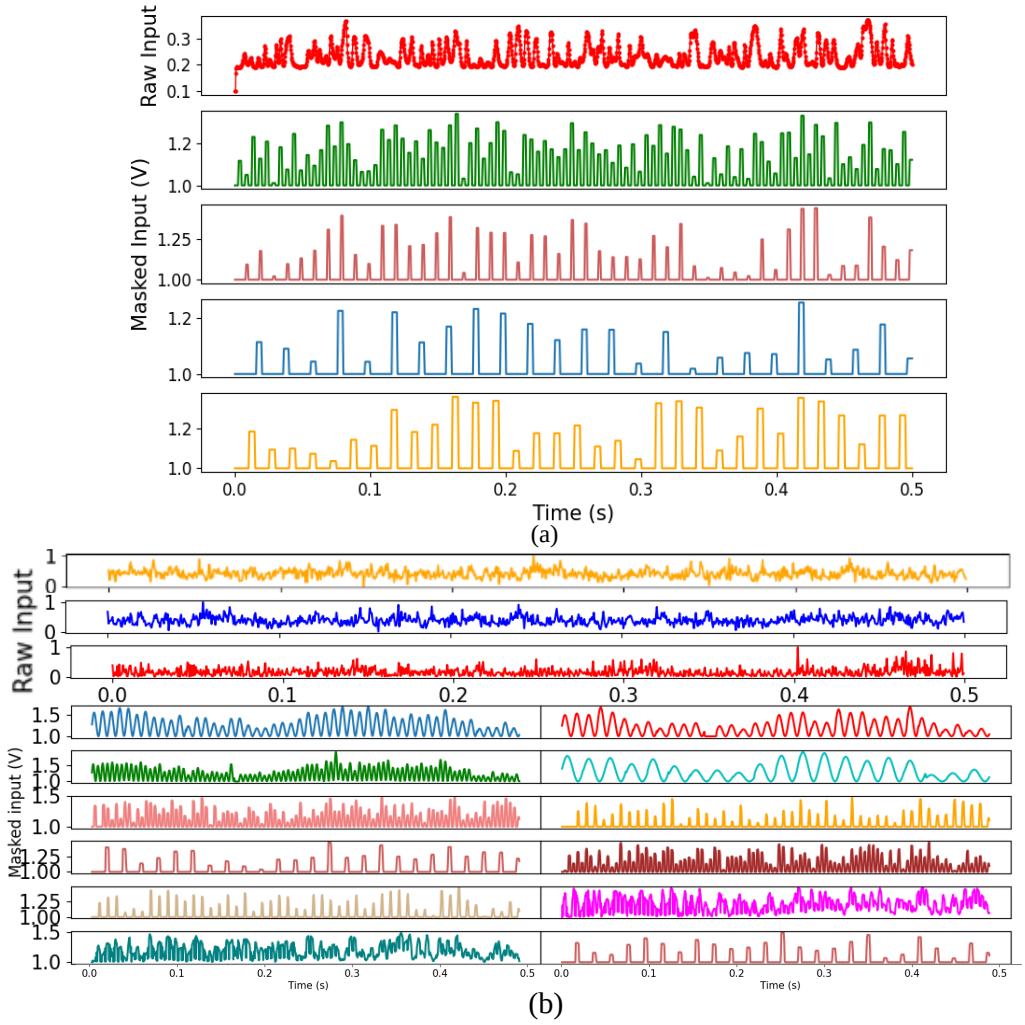


Figure 4.19: The visualization of masked input signals by our proposed approach. (a) Adaptive masked input signals for single prediction task (Narma-20); (b) Adaptive masked input signals for multi-tasking prediction task (Narma-10, Narma-20 and Tree).

The Effect of the Modularity

In order to verify the effect of moduarity achieved by evolving memristor states, we use the multitasking prediction tasks in comparison with the single prediction tasks. Table 4.10 shows the results of the experiments with and without evolving memristor states to single task prediction and multi-tasking prediction, where the one without evolving memristor states are initialized randomly. According to the RMSE, for both the single and multi-tasking prediction, the approach with the evolved memristor states outperforms the one without

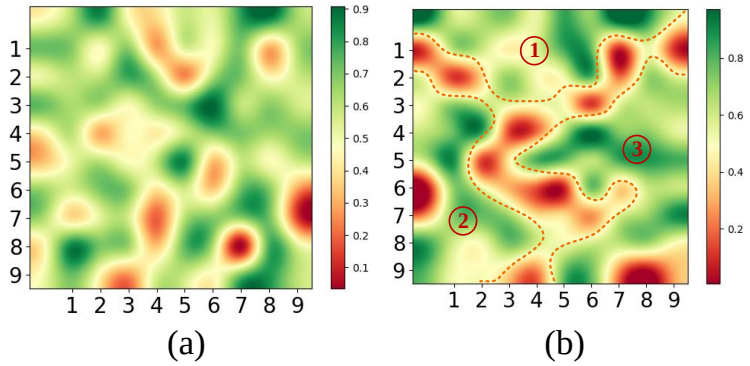


Figure 4.20: The visualization of memristor states under single and multi-tasking prediction task. (a) Single prediction task for Narma-10; (b) Multi-tasking prediction task for Narma-10, Narma-20 and Nonlinear audio.

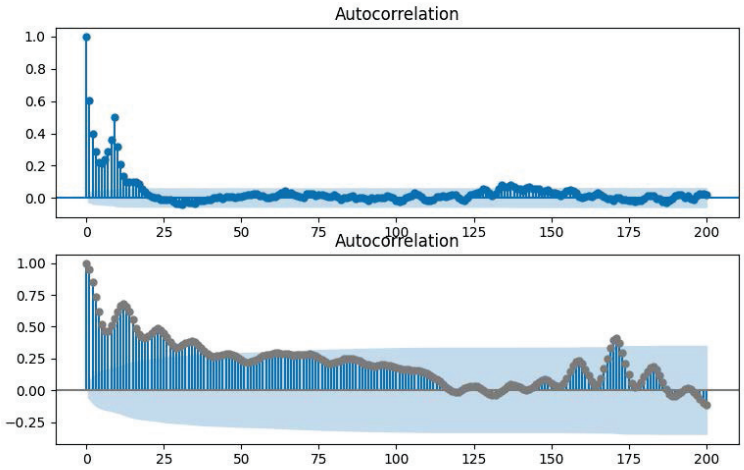


Figure 4.21: Autocorrelation plot of Narma-10 (top) and Nonlinear audio dataset (bottom).

the evolved memristor states. According to Table 4.10, the multi-tasking prediction will not cause performance degradation for those of the sub-task with longer-dependency tasks. Moreover, the results shown in Table 4.10 also indicate that there will be greater degradation of predictive performance for the multitasking prediction tasks when the evolution of the memristor states is ablated, compared with the one for the single prediction task. These results suggest that the multitasking prediction tasks rely more on the evolved modularity.

We visualize the memristor states under single and multi-tasking prediction tasks in Figure 4.20. The red area means that the memristance located here is larger. Therefore, the

memritaor states evolved under the single Narma-10 prediction task incur less modularity distribution (see Figure 4.20 (a)). In Figure 4.20 (b), the evolved memristor states under the multi-tasking prediction task exhibit more modularity distribution, where the red area divides the overall memristor-based crossbar into three parts naturally. It indicates that the multi-tasking prediction tasks tend to result in reservoir with modularity during the evolution.

We also analyze the interactions of the sub-tasks of the multitasking prediction. According to Equations (4.16) and (4.17), Narma-10 is a prediction task of recalling the previous 10 lags, which relies on a shorter-term memory, while Narma-20 will rely on a longer-term memory with 20 lags. As for Nonlinear Audio task and Tree task, their memory dependency is even larger. The auto-correlation plots can depict the relations [354], which are shown in Figure 4.21. We can see that the Nonlinear Audio task has longer-dependency memory compared with Narma-10 task. Taking the dataset of Narma-10 as an example, we can see that the predictive performance of Narma-10 under the multitasking prediction is better than that under carrying on the single task (shown in Table 4.10). Therefore, when the short-term memory dataset Narma-10 was processed together with other long-term memory datasets (Narma-20 or Nonlinear Audio), its predictive performance was improved by the extended memory dependency brought by evolving Narma-20 and Nonlinear Audio datasets with a longer-dependency memory.

In summary, we investigate the effect of adaptive mask and the modularity to the performance of proposed evolutionary algorithm. Specifically, the evolution of adaptive mask led to better RMSE and a more noise-tolerant memristive crossbar-based reservoir. Moreover, the evolution of memristor states enabled the modularity of memristive crossbar-based reservoir, which is particularly beneficial to process multi-tasking prediction tasks.

4.4.4 Comparisons With Existing Reservoir Models

In this section, we will further investigate that if the sneak currents can be used to construct the memristive reservoir states through our evolutionary approach, and to what extent this can lead to good performance. RNN and ESN will be taken as the baseline approaches to verify if the evolved memristive crossbar can behave as a reservoir for time series prediction. Besides these software reservoirs, we also compare our evolved reservoirs with some existing reservoir circuits.

We compare the Root Mean Squared Error (RMSE) results to evaluate the predictive performance of the approaches, and discuss these results in view of certain features of the reservoirs such as the type of architecture, topology, speed of information processing and number of reservoir states. The RMSE is computed based on the test data as follows [9]:

$$RMSE = \sqrt{\langle \|\hat{\mathbf{y}}(t) - \mathbf{y}(t)\|^2 \rangle}, \tag{4.18}$$

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 means. The comparison of RMSE is based on the average RMSE over 10 runs and is supported by Mann–Whitney U tests tests across problems to compare the ranking of the approaches, with level of significance of 0.05.

The comparison results are listed in Table 4.11. In terms of RMSE results, our evolved memristive reservoir circuits with 64 and 100 hidden nodes outperformed the corresponding software reservoirs ESN-64 and RNN-64, and ESN-100 and RNN-100, respectively, when predicting Narma-10, Narma-20, and Tree datasets. Our evolved memristive crossbar with 64 and 100 nodes outperformed the existing hardware reservoirs SCR and CRJ with 64 and 100 nodes, respectively, on all datasets. This was the case both when using our approach to learn a single task or to perform multiple tasks concurrently. Our approach with 100 nodes

obtained better RMSE than our approach with 64 nodes, showing that a larger number of nodes can help to improve predictive performance. Moreover, we also compare the prediction results of evolving reservoir based on 4T1R architecture (our previous work [260]) and evolving reservoir based on 1R architecture. As the results shown in Table 4.11, our previous work [259] outperforms other approaches in terms of the average ranking. Benefiting from the 4T1R-based architecture introduced in Section 4.2.1, the current flowing from the memristor could be either positive or negative during the information processing, therefore, the dynamics of the reservoir could be dynamic to either enhance or decay the memory. Although 4T1R-based work outperforms our proposed 1R-based approach in terms of the prediction performance, 4T1R work is based on more devices with higher circuit overhead, which will be discussed detailed in Section 4.4.5.

When comparing the ranking of the approaches in terms of RMSE across tasks, the Mann-Whitney U tests show that our approach significantly outperformed all the other software and hardware implementations. The evolvable topology of our approach compared to the fixed topology of the other approaches may have played a role on its positive results. In addition, our approach learning multiple tasks concurrently outperformed the approach learning tasks in isolation. These results support our hypothesis that the proposed approach is particularly beneficial for multitasking. A more detailed analysis of our proposed approach, including of the role of modularity on its positive results, has been discussed in Section 4.4.3.

It is worth noting that, in addition to the positive results in terms of RMSE, our memristive crossbar can perform three tasks synchronously (Multitask-64 and Multitask-100), which means that the speed of its information processing is almost 3 times faster than that of other approaches. In addition, compared with memristor-based specific designed circuit, our evolved circuit is based on a 1R memrsitve crossbar, which is a regular architecture without any specific designed structure, which is a desirable property for hardware implementation, as it is easier and more cost-efficient to be fabricated. In particular, the

1R memristor crossbar is a general purpose piece of hardware that is already used for other purposes such as weight mapping or data storage. Here, we show that this piece of hardware can conveniently also be used for reservoir computing.

4.4.5 Evolved Circuit Performance Comparisons

In this section, we will compare our evolved memristive crossbar-based reservoir circuit with other hardware reservoirs in terms of resource consumption.

We can estimate the maximum power consumption of the 1R memristive crossbar as $P_{MR} = V_{max} \times G_{max} \times N_{mem} \times t_r \times f_s = 0.6\mu W$, where N_{mem}, t_r and f_s are the number of memristor devices used in this task ($N_{mem} = 10 \times 10$), the pulse width of the input voltage ($t_r = 10\mu s$) and the max sampling rate of the input waveform ($f_s = 30Hz$), respectively. We also estimate the area overhead of our proposed circuit as $Area_{mem} = 4f^2 \times N_{mem} = 1.52\mu m^2$, where f denotes the half-pitch (the half of the minimum distance between two metal wires).

Table 4.12 shows the comparisons of different hardware reservoirs. We can see that ESN or LSM models have to use n neurons to build the reservoir; however, one neuron always requires several Mosfets to simulate nonlinear behavior between the input and output signals. Taking FPGA-based ESN [344] as an example, 37 Mosfets and 1 capacitor are required to construct one neuron, and more than 60 neurons would be used to construct a reservoir in their work, resulting in 2368 Mosfets and 64 capacitors in its circuit. In comparison to these works [344, 158, 238], our approach used only $1n$ memristors to generate varied nonlinear behaviors, alleviating the problem of circuit scalability. This is because, by using our proposed evolutionary approach, a larger design space of synthesizing the memristor-based crossbar could be searched compared with those of hardware reservoirs by manual design. As a result, less circuit area overhead of constructing a reservoir is achieved by our

Table 4.11: Feature and RMSE comparisons with SOTA models and hardware reservoirs

Comparisons		Feature comparison					RMSE comparison				
		Implementation	Regular Cir. Arch.?	Processing speed	#Nodes	Topology	Narma-10	Narma-20	Jazz	Tree	Ave. Ran*.
ESN-64		Software	-	×1	64	Random fixed	0.1713 (17)	0.1924 (15)	0.0345 (3)	0.2181 (11)	14
RNN-64			-	×1	64	Defined fixed	0.0934 (14)	0.2116 (17)	0.0336 (2)	0.2714 (13)	14
ESN-100			-	×1	100	Random fixed	0.0773 (11)	0.1741 (12)	0.0327 (1)	0.2192 (12)	10
RNN-100			-	×1	100	Defined fixed	0.0458 (7)	0.1677 (11)	0.1325 (9)	0.2871 (14)	12
Memristor-CMOS reservoir circuits											
Random circuit		Specific designed Mem+CMOS circuit	No	×1	64	Random fixed	-	-	-	-	-
SCR circuit-64			No	×1	64	SCR fixed	0.1943 (18)	0.2135 (18)	0.1569 (14)	0.1983 (10)	18
CRJ circuit-64			No	×1	64	CRJ fixed	0.1310 (16)	0.2011 (16)	0.1498 (13)	0.1972 (8)	17
SCR circuit-100			No	×1	100	SCR fixed	0.1235 (15)	0.1741 (12)	0.1429 (12)	0.1903 (7)	14
CRJ circuit-100			No	×1	100	CRJ fixed	0.0876 (13)	0.1871 (14)	0.1387 (11)	0.1857 (6)	13
Evolved reservoir circuits by our previous work											
Work proposed in Section 4.2.1 (node 64)		4T1R Memristor crossbar	Yes	×1	64	Evolvable	0.0219 (1)	0.0427 (2)	0.0421 (4)	0.0625 (2)	1
Work proposed in Section 4.2.1 (node 100)			Yes	×1	100	Evolvable	0.0223 (2)	0.0405 (1)	0.0433 (5)	0.0606 (1)	2
Our evolved reservoir circuits											
Singletask-64		1R Memristor crossbar	Yes	×1	64	Evolvable	0.0839 (12)	0.0911 (7)	0.1370 (10)	0.1982 (9)	11
Singletask-100			Yes	×1	100	Evolvable	0.0449 (6)	0.0886 (6)	0.0959 (8)	0.1271 (4)	8
Multitask-64	Narma-10+Narma-20		Yes	×2	64	Evolvable	0.0563 (8)	0.0999 (8)	-	-	6
	Narma-10+Narma-20+Jazz		Yes	×3	64	Evolvable	0.0607 (9)	0.1129 (9)	0.0844 (7)	-	7
	Narma-10+Narma-20+Tree		Yes	×3	64	Evolvable	0.0648 (10)	0.1342 (10)	-	0.1620 (5)	8
Multitask-100	Narma-10+Narma-20		Yes	×2	100	Evolvable	0.0375 (5)	0.0847 (4)	-	-	4
	Narma-10+Narma-20+Jazz		Yes	×3	100	Evolvable	0.0239 (3)	0.0873 (5)	0.0734 (6)	-	5
	Narma-10+Narma-20+Tree		Yes	×3	100	Evolvable	0.0263 (4)	0.0842 (3)	-	0.1197 (3)	3
P-value		Ours VS RNN: 0.0487; Ours VS ESN: 0.0461; Ours VS SCR: 0.0034; Ours VS CRJ: 0.0034; Ours-single VS Ours-multi.: 0.0202 Ours-64 VS Work proposed in Section 4.2.1: 0.0152									

* 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.
* SCR represents the topology of the simple cycle, and CRJ represents the topology of the cycle with a jump.
* The proposed work in Section 4.2.1 has been published [260].

Table 4.12: Comparisons of different hardware reservoirs with our proposed 1R memristor-based reservoir

Methods	Implementation	Reservoir topology	Components used to construct a reservoir		
			#Memristors*	#MosFets	#Capacitors
FPGA-based ESN [344]	FPGA	Random fixed	0	37n*	n
Mosfet crossbar-based ESN [158]	PCB	Random fixed	0	16n	0
CMOS-based LSM [238]	Specific designed circuit	Random fixed	0	24n	3n
Memristor-based reservoir [33]	Specific designed circuit	SCR fixed	1n	0	0
Memristor-based reservoir [286]	Specific designed circuit	SCR/Random fixed	1n	0	0
Previous work proposed in Section 4.2.1	4T1R memristor crossbar	Evolvable	1n	4n	0
Ours	1R Memristor crossbar	Evolvable	1n	0	0

* n represents the number of neurons existing in a reservoir.
* #Memristors, #Mosfets, and #Capacitors refer to the number of memoristor, Mosfets, and capacitors.
* The predictive performance comparison of memristor-based reservoirs with different topologies have been given in Table 4.11.

proposed evolutionary approach.

In terms of performance and resource consumption, our proposed memristive reservoir circuit outperforms the baseline approaches of RNN and ESN in terms of average RMSE ranking across different regression tasks, while using fewer nodes than RNN and ESN. Furthermore, when compared to other existing fixed-topology memristive reservoir circuits, our memristive reservoir circuits not only achieved superior RMSE, but are also more compact in terms of component count than other circuit implementations.

4.5 Summary and Discussion

This chapter introduced two indirect circuit representations and their corresponding evolutionary operations for the memristive reconfigurable architecture based on 4T1R and 1R memristive arrays, respectively. This chapter answers the RQ2, which is *how to evolve memristive circuits based on reconfigurable architectures?*

The contributions of this chapter include the following aspects:

- In terms of hardware, we designed memristive reservoir circuits that can be evolved

on-chip in a reconfigurable way based on 4T1R and 1R memristive architectures, respectively.

- In terms of the algorithms, as for the 4T1R-based memristive reconfigurable architecture, we proposed its indirect circuit presentation and corresponding evolutionary operations. Avoiding the variances driven by the differences between the actual and ideal memristors, the configuration signals of memristor are evolved and the current states flowing through the memristors are recorded as reservoir states. In addition, a scalable evolutionary algorithm for evolving the reconfigurable memristive reservoir circuits was proposed. The feasibility and scalability of the circuits were taken into the consideration in the proposed algorithm, where sparse initialization and several evolutionary operators were designed for the memristive reservoir circuits, alleviating these two issues.

- As for 1R-based memristive reconfigurable architecture, we also proposed its indirect circuit presentation and corresponding evolutionary operations. We have studied how to manipulate the sneak currents of 1R memristor-based crossbar for reservoir computing instead of preventing them, where the input connection, input timing, the memristor states and the crossbar size are evolved. In order to harness the sneak currents, an evolutionary algorithm with novel evolutionary operators was proposed to evolve the memristive reservoir spatiotemporally. This evolutionary algorithm provides an automatic tool for synthesising the memristive crossbar's memristor states, input-connection states, and input-injection timing, thus evolving the reservoir's adaptive mask and modularity.

- We conducted experiments to validate our evolved memristive crossbar-based reservoir architectures based on 4T1R arrays. We applied 1 generation task, 1 classification task, and 6 prediction tasks to evaluate the performance of the 4T1R memristive reservoir circuit. The experimental results show that our proposed algorithm can evolve

a memristive reservoir circuit with a better regression performance than the other memristive reservoir with a fixed topology and 5 baseline approaches, as measured by the average ranking.

- We also conducted experiments to validate our evolved memristive crossbar-based reservoir architectures based on 1R arrays, where we apply the proposed algorithm to 4 datasets including single and multitasking prediction tasks. As shown experimentally, the proposed adaptive mask can improve noise tolerance of the memristor-based crossbar, and the evolved modularity has the potential to be particularly useful when using reservoir computing for multitasking prediction problems. Compared to 4 baseline approaches, our proposed method got better predictive performance over all 4 datasets. Compared to 5 existing hardware reservoirs, our proposed 1R memristive reservoir achieved better predictive performance than the memristive reservoir circuits with random, SCR and CRJ topologies, but worse than our 4T1R work proposed in Section 4.2.1. In terms of the circuit performance, our proposed method can evolve a more scalable reservoir circuit with fewer components and a regular architecture than other hardware reservoirs, including the 4T1R memristive reservoir proposed in Section 4.2.1.

Chapter Five

Time Delay Reservoir with Enhanced Memory and Its Memrisitve Implementation[1]

As introduced in Chapter 2, Reservoir Computing (RC) is a promising application field for evolvable hardware. In order to pursue better performance, the intrinsic dynamics of reservoirs should be adaptive to different tasks with different characteristics. An evolvable reservoir may be better suited for a varying environment. The evolvable hardware approach thus provides a great opportunity where reservoirs could be evolved to fit different tasks.

Chapter 4 investigates how to evolve the different memristor-based reconfigurable architectures for the pure-memristive RC, configured with different typologies to fit different tasks. Time Delay Reservoir (TDR) is another promising application field for EHW. First, it

[1]This chapter corresponds to RQ3 in Section 1.1, and is based on our published paper [264] and our paper under review [262]. Specifically, Section 5.1 presents the memory property analysis of conventional TDR proposed in our published paper [264]. Sections 5.2.1, 5.3.3 and 5.3.4 present the approach proposed in our published paper [264], and Section 5.4 gives its experiment part. Sections 5.2.2, 5.3.2 and 5.3.5 present the approach proposed in our under review paper [262], and Section 5.5 presents its experiment part. This footnote, in addition to the statements in Section 1.2, also serves to clarify any detected overlaps between this thesis and my own published papers.

can prevent the connection overhead of neural networks with increasing neurons, leading to potential improvements on energy efficiency. Second, through its dynamic system representation, TDR can also be implemented in hardware by different systems. However, it obtains lower accuracy on the tasks that involve long-term dependency. Therefore, this chapter will focus on answering RQ3:

RQ3: How to improve the accuracy and energy efficiency of EHW for RC?

In order to investigate the aforementioned research question, we first analyze the memory property of the conventional time delay reservoir, digging into the theoretical reason why the conventional TDR has a limited ability to tackle the long-term dependency problem, which is a theoretical motivation for our proposed method, introduced in Section 5.1. Section 5.2 then introduces our proposed time delay reservoirs with enhanced memory. Section 5.3 presents the memristive implementation of our proposed TDR with enhanced memory. Section 5.4 presents the experimental studies of adaptive memristive memory-enhanced TDR. Section 5.5 presents the experimental studies of fully analog memristive memory-enhanced TDR. The summary of this chapter is given in Section 5.6.

5.1 Memory Property of Conventional Time Delay Reservoir

This section presents an analysis of the existing conventional TDR and its memory property, which was used to investigate the weaknesses of existing work and inform our novel developments.

Mathematical modelling with Delay Differential Equations (DDEs) is widely used in various fields such as population dynamics, epidemiology, and neural networks for analyses

and predictions. The time delays in these models account for the relationship between the modelled system's current state and its past history. The delay can be attributed to the duration of certain hidden processes, such as the stages of the life cycle, the time between virus infection and virus production [92]. DDEs can describe real-world systems, where the events are rarely instantaneous. This can be modelled as follows [92]:

$$y'^{(t)} = f(y^{(t)}, y^{(t-\tau_1)}, y^{(t-\tau_2)}, ..., y^{(t-\tau_d)}, t), t \geq t_0, \tag{5.1}$$

where τ_i are the delay terms, and could be constant, or variable as functions of t or even of the state y. Considering that these features are required in RC [288], DDE is a good approach to describe the dynamic behavior of reservoir.

The states of the reservoir in TDR can be described generally by the solutions of the following DDE [92]:

$$\dot{h}^{(t)} = -h^{(t)} + f(h^{(t-\tau)}, x^{(t)}), \tag{5.2}$$

where $h(t)$ refers to the states of the reservoir, f is a nonlinear function, $x(t)$ is the input signal connected to the reservoir, and f refers to a nonlinear function. With delay interval τ, N equidistant points will be separated in time by $\theta = \tau/N$, and these N equidistant points could be regarded as *virtual neurons* being multiplexed in the given time scale. By Euler discretization of Equation (5.2) with integration step θ, the state of i-th reservoir node at time $k\tau$ $h_i^{(k)}$ can be rewritten as:

$$h_i^{(k)} = \frac{1}{1+\theta} h_{i-1}^{(k)} + \frac{\theta}{1+\theta} f(h_i^{(k-1)}, x_i^{(k)}). \tag{5.3}$$

Considering that there is an error between the output $\hat{y}^{(k)}$ and the target $y^{(k)}$ defined as $\varepsilon^{(k)}$, W_{yh} represents the output weights, the procedure of conventional TDR can be formulated

as:

$$\begin{cases} y^{(k)} = W_{yh} h_i^{(k)} + \varepsilon^{(k)} \\ h_i^{(k)} = \frac{1}{1+\theta} h_{i-1}^{(k)} + \frac{\theta}{1+\theta} f(h_i^{(k-1)}, x_i^{(k)}). \end{cases} \tag{5.4}$$

Let $\mathcal{B}$ be the backshift operator, defined as $\mathcal{B}^j x^{(t)} = x^{(t-j)}$ for $j \geq 0$, applicable to all random variables in a time series $x^{(t)}$. Therefore, ${h_{i-1}}^{(k)} = \mathcal{B}h^{(k)}$ and ${h_i}^{(k-1)} = \mathcal{B}^\tau {h_i}^{(k)}$. Equation (5.3) can be rewritten as:

$$h_i^{(k)} = \frac{1}{1+\theta} \mathcal{B} h_i^{(k)} + \frac{\theta}{1+\theta} f(\mathcal{B}^\tau {h_i}^{(k)}, x_i^{(k)}). \tag{5.5}$$

According to the work proposed in [354], without loss of generality, assume that the linear activation and output functions are identity. Therefore, we adapt the same assumption [354] to TDR analysis, thus we can get:

$$h_i^{(k)} = \left[I - \frac{1}{\theta+1}\mathcal{B} - \frac{\theta}{\theta+1}\mathcal{B}^\tau \right]^{-1} \frac{\theta}{\theta+1} x_i^{(k)}. \tag{5.6}$$

The inverse calculation in Equation (5.6) could be decomposed as:

$$\sum_{j=0}^{\infty} \left[-\left(I - \frac{1}{\theta+1}\mathcal{B} \right)^{-1} \left(-\frac{\theta}{\theta+1}\mathcal{B}^\tau \right) \right]^j \left(I - \frac{1}{\theta+1}\mathcal{B} \right)^{-1}. \tag{5.7}$$

The first term in Equation (5.4) could be written as the form of $y^{(k)} = \sum_{j=0}^{\infty} A_j x^{(k-j)} + \varepsilon^{(k)}$, since $\left(I - \frac{1}{\theta+1}\mathcal{B}\right)^{-1} = \sum_{j=0}^{\infty} (\frac{1}{\theta+1})^j \mathcal{B}^j$, we can get:

$$A_j = \left(\frac{1}{\theta+1} \right)^{3j} \left(\frac{\theta}{\theta+1} \right)^{j+1} W_{yh}, \tag{5.8}$$

A_j will decay exponentially. Therefore, conventional TDR has limited capability of handling long-range dependence data due to this exponential decay.

5.2 Time Delay Reservoir with Memory Enhancement

In this section, we introduce two novel TDRs with memory enhancements. Section 5.2.1 introduces the memory-enhanced TDR through the reservoir states, and Section 5.2.2 presents the memory-enhanced TDR through the input masking. The training algorithm of our proposed TDR's readout layer is introduced in Section 5.2.3, and the optimization of our proposed memory-enhanced TDR is proposed in Section 5.2.4, the memory property analysis of our proposed memory-enhanced TDR is given in Section 5.2.5.

5.2.1 Memory Enhancement Through Reservoir States

Let $\{x^{(t)}\}$, $\{\hat{y}^{(t)}\}$, $\{y^{(t)}\}$ be the input, output, and target sequences of a time series, respectively, where $\hat{y}^{(t)} \in \mathbb{R}^p$, $y^{(t)} \in \mathbb{R}^p$. The enhanced memory is introduced to TDR by a higher-order delay unit, which can be depicted as:

$$D\left(h^{(t)};\lambda\right) = \left[\left((I-\mathcal{B})^{\tau+\lambda} - I\right)\right] h^{(t)}, \tag{5.9}$$

where $\mathcal{B}$ is the backshift operator, $\lambda = (\lambda_1, ..., \lambda_m)'$, and $\lambda \in [0, 1]$ indicate the enhanced degree, $h(t)$ represents the reservoir states, and τ is the delay interval. Therefore, the reservoir states can be described by the solution of the following equation:

$$\dot{h}^{(t)} = -h^{(t)} + f\left(h^{(t-\tau)}, D\left(h^{(t)};\lambda\right), x^{(t)}\right). \tag{5.10}$$

The higher-order delay unit $D\left(h^{(t)};\lambda\right)$ can accumulate the previous m states from long-term history, which will provide the enhanced memory for TDR. As λ could be 0, the related state in the j-th layer could not be accumulated to the current state. Formally, as

for the i-th higher-order delay unit, it has:

$$D\left(h^{(t)};\lambda\right)_i = \sum_{j=1}^{m}\left[h_i^{(t-(j+\lambda_j)\tau)}\right]. \tag{5.11}$$

With delay interval τ, N equidistant points will be separated in time by $\theta = \tau/N$, and these N equidistant points could be regarded as *virtual neurons* being multiplexed in the given time scale. By Euler discretization of Equation (5.10) with integration step θ, the procedure of memory-enhanced TDR could be formulated as:

$$\begin{cases} y^{(k)} = W_{yh}h_i^{(k)} + \varepsilon^{(k)} \\ h_i^{(k)} = \frac{1}{\theta+1}h_{i-1}^{(k)} + \frac{\theta}{\theta+1}f\left(h_i^{(k-1)}, D\left(h^{(k)};\lambda\right)_i, x^{(k)}\right) \\ D\left(h^{(k)};\lambda\right)_i = \sum_{j=1}^{m}\left[h_{\lambda_j\cdot\tau}^{(k-j)}\right], \end{cases} \tag{5.12}$$

where f is a nonlinear function, and there is the error $\varepsilon^{(k)}$ between the output $\hat{y}^{(k)}$ and the target $y^{(k)}$. The error is a sequence of random vectors that are independent and have the same distribution over time. We also illustrate the procedure of Equation (5.12) in Figure 5.1. As shown in Figure 5.1, within an interval τ, the TDR is discretized as N *virtual neurons* in the vertical direction. These *virtual neurons* are all history-dependent, so that history states could be transferred in the horizontal direction. In addition, the neuron states in the long-term history can also be transferred to the current state from the higher-order unit $D\left(h^{(k)};\lambda\right)_i$. Therefore, the current reservoir state $h_i^{(k)}$ in the memory-enhanced TDR is traced from three parts:

- Neighboring-dependency in the same layer $h_{i-1}^{(k)}$: the state of its closest neighbourhood in the same layer will be transferred to the current state.
- Self-dependency in the previous layer $h_i^{(k-1)}$: the self inertance of states in the previous layer will be considered to the current state;

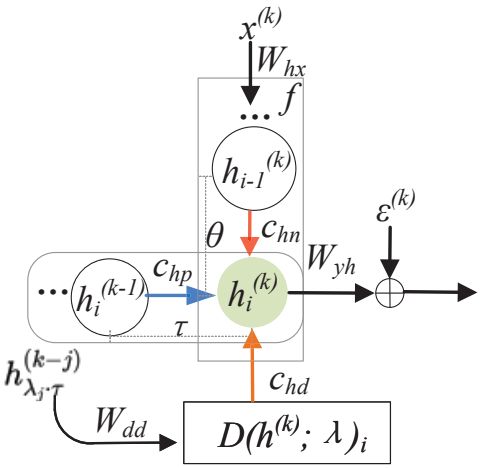


Figure 5.1: Reservoir state of memory-enhanced TDR. $W_{hx} \in \mathbb{R}^{p \times q}$ and $W_{yh} \in \mathbb{R}^{p \times q}$ represent the input weights and output weights respectively; c_{hp}, c_{hn}, and c_{hd} represent the transfer factors between self-dependency, neighboring-dependency and long-term dependency to current states, respectively.

- Long-term dependency $\sum_{j=1}^{m} \left[h_{\lambda_j \cdot \tau}^{(k-j)} \right]$ from previous m layers: the states from long-term history will be accumulated to present states.

Figure 5.2 shows the discrete form of our proposed memory-enhanced TDR, which specifically illustrates how the higher-order delay unit establishes the long-term dependency to enhance the memory of TDR. For the previous j-th layer ($j \in (1, ..., m)$), there will be the corresponding state $h_{\lambda_j \cdot \tau}^{k-j}$ related to current reservoir state $h_i^{(k)}$, where $\lambda = (\lambda_1, \lambda_j, ...\lambda_m)$ indicates which state in the previous m layers will be selected. Considering the large search space incurred by the length of time sequence and delayed states, we will apply PSO algorithm to determine which specific delayed states should be transferred to the current reservoir state for different tasks. The details of this will be introduced in Section 5.2.4.

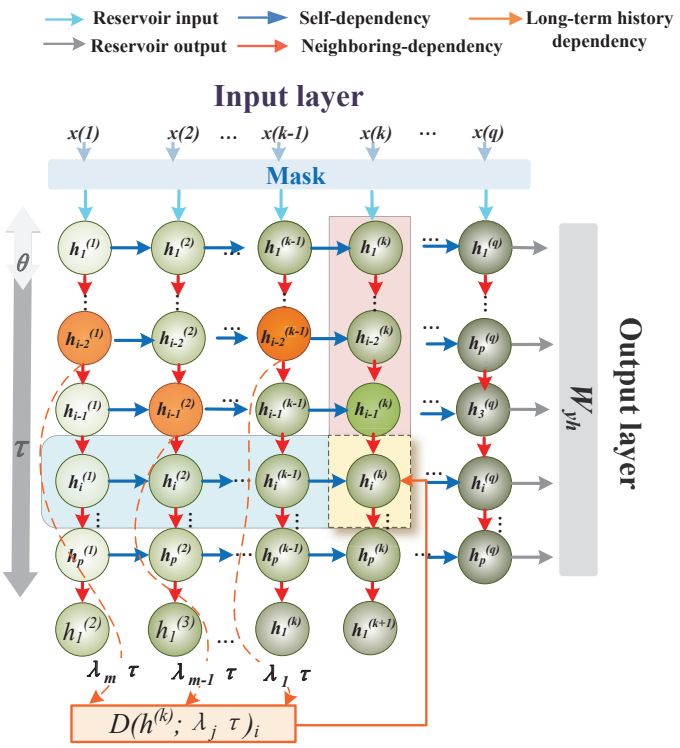


Figure 5.2: The discrete form of our proposed memory-enhanced TDR.

5.2.2 Memory Enhancement Through Input Masking

The dynamics of TDR can be described through a DDE [92]:

$$\dot{h}^{(t)} = -h^{(t)} + f(h^{(t-\tau)}, x^{(t)}), \tag{5.13}$$

where f is a nonlinear function and we can see that the memory property of a TDR will not only depend on the reservoir states $h^{(t)}$ but also on the input signals $x^{(t)}$. During the processing of the input signal, an issue needs to be addressed, namely the lack of interaction between neighboring time steps, which results in missing the short-term information existing in the neighboring time steps. To address this issue, we propose a dynamic mask that incorporates the treatment of neighboring relevance. Specifically, we propose using a moving window to enhance the feature values of neighboring time steps, taking into account the

dynamic nature of the data. This allows us to incorporate neighboring time dependencies in a more effective manner.

Let $\mathbf{z}^{(t)} \in \mathbb{R}^n, t \in \mathbb{Z}$ be an n-dimensional time input signal. This signal is, first, time and dimensionally multiplexed over a delay period by using an input mask $\mathbf{C} \in [0,1]^{N\times n}$, where N is a design parameter called the number of neurons of each reservoir layer. The multiplexed discrete-time N-dimensional signal $\mathbf{C}\mathbf{z}^{(t)}, t \in \mathbb{Z}$ is a masked input. We propose a dynamic mask involving the neighboring history of the input features, which can be described as follows:

$$\mathbf{x}^{(t)}(\mathbf{z}, \mathbf{C}, \boldsymbol{\mu}) = \mathbf{c}^{(t)} \cdot \mathbf{z}^{(t)} + (1 - \mathbf{c}^{(t)}) \cdot \boldsymbol{\mu}^{(t;W)}, \tag{5.14}$$

where W refers to the relevance parameter that controls how the moving window depends on neighboring times. By Euler discretization to the continuous-time sequence, we can get:

$$\mathbf{x}_i^{(k)} = \mathbf{c}_i^{(k)} \cdot \mathbf{z}_i^{(k)} + (1 - \mathbf{c}_i^{(k)}) \cdot \boldsymbol{\mu}_i^{(k;W)}, \tag{5.15}$$

where $\mu_i^{(k;W)}$ is defined as:

$$\boldsymbol{\mu}_i^{(k;W)} = \frac{1}{1+W} \sum_{j=k-W}^{k} \mathcal{B}^j \mathbf{z}_i^{(k)}, \tag{5.16}$$

where $\mathcal{B}^j$ is the backshift operator, defined as $\mathcal{B}^j \mathbf{z}^{(k)} = \mathbf{z}^{(k-j)}$ for $j \geq 0$.

We also illustrate the aforementioned procedures of dynamic masking in Figure 5.3 (a). As shown in Figure 5.3 (a), $\boldsymbol{\mu}^{(k)}$ represents the signals after the average sliding window operation to $\mathbf{z}$. Then the signal $\boldsymbol{\mu}$ will be weighted with $1 - \mathbf{c}^{(k)}$ and accumulated to the signal $\mathbf{z}$ weighted with $\mathbf{c}^{(k)}$. So that we can get the masked input $\mathbf{x}^{(k)}$. By building this dynamic mask, the dynamic nature of the data could be exploited, which is crucial if we want to capture local time variations of the features.

Masking is described as a technique that enhances the temporal interaction within an

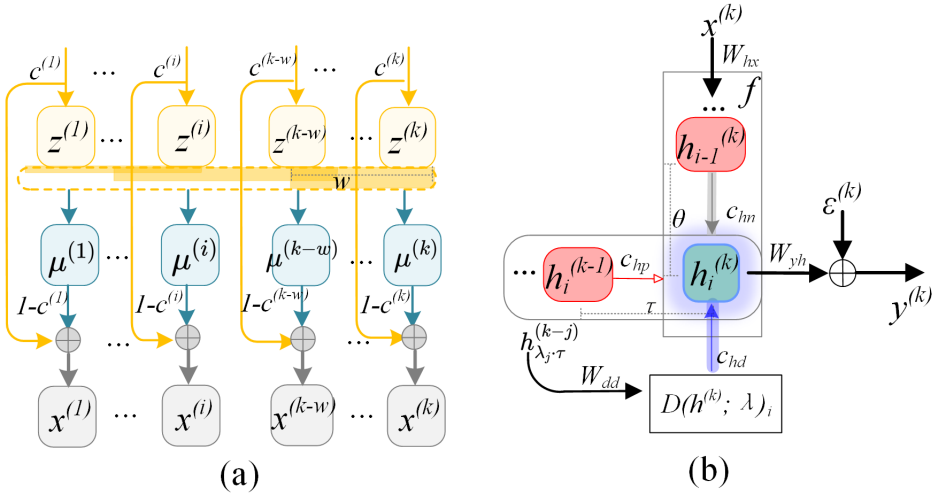


Figure 5.3: (a) The discrete form of our proposed dynamic masking technique. (b) The discrete form of our proposed state forwarding technique.

input signal. This is achieved through the use of a moving window mechanism, which operates in conjunction with a memristor-based crossbar architecture. The process of masking is specifically conducted within the dynamic input layer of the fully analog memristive TDR. As the input signal enters the dynamic layer, it is first sampled and held. This sampled signal is then weighted by the memristor-based crossbars, which are capable of adjusting the strength of the signal based on the memristive properties. Finally, the weighted signals are summed together to produce the masked input. This masked input is what is then fed into the reservoir for further processing.

5.2.3 Training Algorithm of The Readout Layer

The TDR's readout layer can be trained both offline and online by minimizing a give loss function. In most cases, we can evaluate the model performance via the root mean square error (RMSE), which could be described as follows:

$$RMSE = \sqrt{\langle \|\hat{y}(t) - y(t)\|^2 \rangle}, \tag{5.17}$$

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

Offline Training

The readout weights W_{yh} can be trained by offline mode. With the input signal of reservoir $x(t)$, there is a corresponding teaching signal $y(t) \in \mathbb{R}^p$ and a p-dimensional output could be obtained by output matrix and reservoir state $y(t) := h(t)^{\mathsf{T}} \cdot W_{yh}$. The training process will find the output weights $W_{yh} \in \mathbb{R}^{p \times q}$ by minimizing the distance between the output and the teaching signal, which is described as the following optimization problem:

$$W_{yh} := \arg\min_{W} \left(\sum_{i=1}^{M} ||{h_i^{(t)}}^{\mathsf{T}} \times W - y_i^{(t)}||^2 + \eta ||W||^2 \right) \tag{5.18}$$

where $||W||^2$ refers to a regularisation term to prevent overfitting, and η controls its intensity. In order to optimize this problem, ridge regression [113] has been applied, whose solution could be given by:

$$W_{yh} = (HH^{\mathsf{T}} + \eta H)^{-1} Hy. \tag{5.19}$$

Online Training

Using standard numerical optimization methods, the Levenberg–Marquardt (LM) algorithm is a fast technique that combines the local convergence of the Gauss-Newton method and the global properties of the gradient descent method. The LM algorithm first searches in the direction opposite to the gradient, and then near the optimum, it produces a new improved search direction based on the Newton method [163]. The following steps describe how the LM algorithm works:

•**Step 1:** Initialize the weights of the output layer W_0.

•**Step 2:** Set the iteration counter k to 0.

•**Step 3:** Compute the Jacobian matrix $\boldsymbol{J}_k$ and the residual vector r_k using the current weights W_k. J_k is the Jacobian matrix (a matrix of first partial derivatives of a function), r_k is the residual vector (the difference between the model's predictions and the observed data).

•**Step 4:** Compute the update step ΔW_k using the following equation:

$$\Delta W_k = -(J_k^T J_k + \sigma_k I)^{-1} J_k^T r_k \tag{5.20}$$

where I is the identity matrix and σ_k is a damping factor.

•**Step 5:** Update the parameters of the model using the following equation:

$$W_{k+1} = W_k + \Delta W_k \tag{5.21}$$

•**Step 6:** Increment the iteration counter k by 1.

•**Step 7:** Check if the stopping criterion has been met. If it has, terminate the algorithm and return the final weight values W_k. Otherwise, go to step 3 and repeat the process.

The LM algorithm iteratively updates the parameters of the model until the residual vector is minimized or until a predetermined stopping criterion is met. At each iteration, the algorithm computes the Jacobian matrix and the residual vector, and then uses these quantities to compute the update step. The update step is then used to update the parameters of the model, and the process is repeated until convergence is achieved.

5.2.4 Evolving Better Memory for TDR

Particle Swarm Optimization (PSO), one of the famous swarm intelligence optimization algorithms, searches for optimal solutions using a population of particles [59]. Each particle, marked by a pair of position and velocity $(\boldsymbol{p_i}, \boldsymbol{v_i})$, represents a candidate solution to the given problem. Let $p_{i_{best}}$ denote the personal best position of the ith particle, and g_{best} denote the global best position among all particles so far in the search process. The objective function to be optimised is depicted by a function called the fitness function. The velocity and the position of each particle in a search space can be iteratively updated with the following equations:

$$\begin{aligned} \boldsymbol{v_i}^{(t+1)} = w \cdot \boldsymbol{v_i}^{(t)} + c_1 \cdot rand() \cdot (\boldsymbol{p}_{i_{best}} - \boldsymbol{p_i}^{(t)}) \\ + c_2 \cdot rand() \cdot (g_{best} - \boldsymbol{p_i}^{(t)}), \end{aligned} \tag{5.22}$$

$$\boldsymbol{p_i}^{(t+1)} = \boldsymbol{p_i} + \boldsymbol{v_i}^{(t+1)}, \tag{5.23}$$

where w presents the inertia weight to balance between exploration and exploitation process, c_1 and c_2 are factors used to moderate the displacements of particles toward the local or the global optimum, and $rand()$ is a random function in the range $[0, 1]$.

Algorithm 5.1 gives the pseudo code showing how PSO is applied to enhance the memory of TDR. This algorithm receives as input the population size N, the maximum possible order m for the higher-order delay unit, the maximum number of iterations max_Iter and the fitness function f. The fitness function can potentially be any measure of predictive performance of the reservoir. In our experiments, we will use Root Mean Squared Error (RMSE), which will be explained in Section 5.4.2. Each particle i's position $\boldsymbol{p_i}$ corresponds to a candidate value for $\lambda = (\lambda_1, \lambda_j, ..., \lambda_m)'$. We first initialize the population of particles with N particles corresponding to λ values picked uniformly at random from $[0, 1]^m$.

Then, if the number of iterations has not reached the pre-defined value of max_Iter,

Algorithm 5.1 Pseudo code of PSO optimization of the degree of memory enhancement for TDR

Input: population_Num: N, order_Num: m, max_Iter, fitnessFunction: f
Output: $\boldsymbol{p\_global\_best}$

1: Initialise population $\boldsymbol{p}$
2: **while** max_Iter not met **do**
3: **for** each individual $\boldsymbol{p_i}$ in $\boldsymbol{p}$ **do**
4: Generate memory enhanced TDR with $\boldsymbol{p_i}$;
5: Compute velocity $\boldsymbol{v_i}$of each individual by Equation (5.22);
6: Update individual $\boldsymbol{p_i}$ by Equation (5.23);
7: Calculate $fitness_i$ of individual $\boldsymbol{p_i}$ by f;
8: **end for**
9: **Set:** $local\_best\_fitness \longleftarrow max(\{fitness_i\})$
10: **if** $local\_best\_fitness > global\_best\_fitness$ **then**
11: $\boldsymbol{p\_global\_best} \longleftarrow \boldsymbol{p\_local\_best}$
12: $global\_best\_fitness \longleftarrow local\_best\_fitness$
13: **end if**
14: **end while**
15: **Return:** $\boldsymbol{p\_global\_best}$

the following steps will be executed (Ln 2-Ln 14). A memory-enhanced TDR will be generated for each individual $\boldsymbol{p_i}$. Next, the velocity $\boldsymbol{v_i}$ of each particle will be computed by Equation (5.22), and particle $\boldsymbol{p_i}$ will be updated by Equation (5.23). Then, the fitness of $\boldsymbol{p_i}$ will be calculated by fitness function f. After that, we will update the $local\_best\_fitness$ by the max fitness in the population. Next, if $local\_best\_fitness > global\_best\_fitness$, $\boldsymbol{p\_global\_best}$ will be updated by $\boldsymbol{p\_local\_best}$, and $global\_best\_fitness$ will be updated by $local\_best\_fitness$. Finally, the $\boldsymbol{p\_global\_best}$ will be returned. This value is the optimised λ to be used in the reservoir.

5.2.5 Memory Property Analysis of Our Proposed TDR

In this section, we provide a theoretical analysis of the memory properties of the proposed memory-enhanced TDR approach. We use a process of our proposed TDR with n-dimension input ($n = 1$) as an example to illustrate the memory property derivation. Without loss

of generality, assume that the linear activation and output functions are identified. By discretizing Equation (5.10) first, we can rewrite it as:

$$h_i^{(k)} = \left[I - \tfrac{1}{\theta+1}\mathcal{B} - \tfrac{\theta}{\theta+1}\mathcal{B}^{\tau} - \tfrac{\theta}{\theta+1}\sum_{j=1}^{m}\mathcal{B}^{(\lambda_j+j)\tau}\right]^{-1} \cdot \tfrac{\theta}{\theta+1}x_i^{(k)}. \tag{5.24}$$

Since $\left(I - \frac{1}{\theta+1}\mathcal{B}\right)^{-1} = \sum_{j=0}^{\infty}(\frac{1}{\theta+1})^j\mathcal{B}^j$, we could obtain:

$$h_i^{(k)} = \sum_{j=0}^{\infty}\left[-(I - \tfrac{1}{\theta+1}\mathcal{B})^{-1}\tfrac{\theta}{\theta+1}(\mathcal{B}^{\tau} + \sum_{j=1}^{m}\mathcal{B}^{(\lambda_j+j)\tau})\right]^j \cdot (I - \tfrac{1}{\theta+1}\mathcal{B})^{-1}\tfrac{\theta}{\theta+1}x_i^{(k)}. \tag{5.25}$$

Since $\sum_{j=0}^{\infty}(\frac{1}{\theta+1})^j\mathcal{B}^j = ((I-\mathcal{B})^{(\lambda_j+j)\tau} - I)$, then Equation (5.25) can be further transformed into the following form:

$$h_i^{(k)} = \sum_{j=0}^{\infty}\left[-(I - \tfrac{1}{\theta+1}\mathcal{B})^{-1}\tfrac{\theta}{\theta+1}[\mathcal{B}^{\tau} + ((I-\mathcal{B})^{(\lambda_j+j)\tau} - I)]\right]^j \cdot (I - \tfrac{1}{\theta+1}\mathcal{B})^{-1}\tfrac{\theta}{\theta+1}x_i^{(k)}. \tag{5.26}$$

Let $y^{(k)} = \sum_{j=0}^{\infty}A_j x^{(k-j)} + \varepsilon^{(k)}$, then $A_j = C_j + D_j$, where

$$\begin{cases} \sum_{j=0}^{\infty}C_j = [(I - \frac{1}{\theta+1}\mathcal{B})^{-1}\frac{\theta}{\theta+1}]\mathcal{B}^{\tau}(I - \frac{1}{\theta+1}\mathcal{B})^{-1}x_i^{(k)} \\ \sum_{j=0}^{\infty}D_j = [(I - \frac{1}{\theta+1}\mathcal{B})^{-1}\frac{\theta}{\theta+1}][(I-\mathcal{B})^{(\lambda_j+j)\tau} - I] \cdot (I - \frac{1}{\theta+1}\mathcal{B})^{-1}x_i^{(k)} \end{cases} \tag{5.27}$$

From the analysis of conventional TDR proposed in [264], the entries in C_j decay exponentially, as well as the entries in the first part $[(I - \frac{1}{\theta+1}\mathcal{B})^{-1}\frac{\theta}{\theta+1}]$ in D_j. Since $[(I-\mathcal{B})^{(\lambda_j+j)\tau} - I] = \sum_{j=0}^{\infty}W_k\mathcal{B}^k$ and W_k's are diagonal matrices with $(W_k)_{ii} \sim k^{-(\lambda+j)\tau-1}$, the decay of D_j is dominated by $[(I-\mathcal{B})^{(\lambda_j+j)\tau} - I]$. According to [89], as $j \to \infty$, entries of A_j time series exhibit longer memory than the one provided in work [264]. When we take the effect of dynamic masking into consideration, we can substitute $x_i^{(k)}$ in Equa-

tion (5.26) by Equation (5.15). Since the right term of Equation (5.15) dominates the equation, Equation (5.26) could be approximated by Equation (5.28). As for the dynamic masking, $j \in [k-W, W]$ in this component of the equation can summation will only go through the most recent W entries, whereas the effect of the state forwarding go much further into the past ($j \to \infty$). It will decay at the rate $\frac{1-c^{(k)}}{1+W}$ when $j \to \infty$, which will decay quicker than the part of state forwarding in Equation (5.28), thus this component of the equation will enhance the short-term information that existed in the past W steps of the time series.

Based on this analysis, the technique of dynamic masking enhances the memory that existed in the neighboring time steps, whereas the technique of state forwarding creates a longer memory than the conventional TDR.

$$h_i^{(k)} = \underbrace{\sum_{j=0}^{\infty} \left[-(I - \frac{1}{\theta+1}\mathcal{B})^{-1} \frac{\theta}{\theta+1}[\mathcal{B}^{\tau} + ((I-\mathcal{B})^{(\lambda_j+j)\tau} - I)]\right]^j \cdot \left[(I - \frac{1}{\theta+1}\mathcal{B})^{-1} \frac{\theta}{\theta+1}\right]}_{\text{state forwarding}} \cdot \underbrace{\left[(1-c_i^{(k)}) \frac{1}{1+W} \sum_{j=k-W}^{k} \mathcal{B}^j z_i^{(k)}.\right]}_{\text{Dynamic masking}} \tag{5.28}$$

5.3 Memrisitve Implementation of Time Delay Reservoir with Memory Enhancement

Considering the hardware-friendly features of TDR and the benefits of memristor, we provide a memristive implementation of the proposed adaptive memory-enhanced TDR and a fully analog memristive TDR in this section. An overview of the system is given in Figure 5.11. It consists of four main blocks: the dynamic input masking module (DIM), the reservoir layer, the output layer, and the particle swarm optimization (PSO). The DIM optimizes the interaction with input signals by applying dynamic masks. These masks are fine-tuned using Particle Swarm Optimization (PSO) techniques, which will be explained detailed in

Section 5.3.2. The reservoir layer of the system incorporates dynamic memristors, which enhance memory capabilities by effectively transferring past states to forward processing as introduced in Section 5.3.3. We will introduce the adaptive memory enhanced memristive TDR in Section 5.3.4. The output layer uses nonvolatile memristors for readout, trained with the Levenberg-Marquardt algorithm to minimize errors in Section 5.2.3. The PSO module harmonizes the DIM and reservoir layer, ensuring optimal parameter settings for diverse tasks and datasets, culminating in a system adept at sophisticated signal processing and adaptation. The PSO module is explained in Section 5.2.4. A summary of the working steps of the system will be introduced in Section 5.3.5.

5.3.1 Memristor Models

Dynamic Memristor Model

Tanaka et al. [288] proposed that some of the memristive devices or systems can be used to build up reservoirs, since they are capable of exhibiting nonlinear dynamic behavior. To construct the nonlinear dynamic feature for the reservoir, we utilized a dynamic memristor model [357] that incorporates a fading memory effect. The reason for selecting this model is its characteristic of volatility. Its mathematical expression is provided as follows:

$$I = KGV^3, \tag{5.29}$$

$$G^{(t)} = G_0 + r(G^{(t-1)} - G_0) + \frac{a\,|V|}{a\,|V| + 1}(G_{th} - G^{(t)}), \tag{5.30}$$

$$\begin{cases} K = sign^{(V)}K_p + sign^{(-V)}K_n \\ G_{th} = sign^{(V)}, \end{cases} \tag{5.31}$$

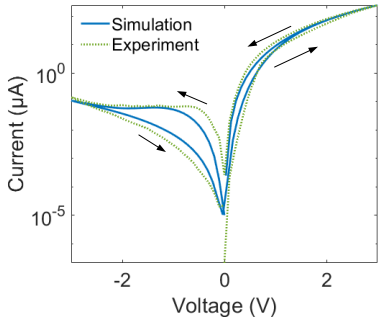


Figure 5.4: V–I hysteresis curves of $Ti/TiO_x/TaO_y/Pt$-based dynamic memristor model, where the experimental data comes from [357].

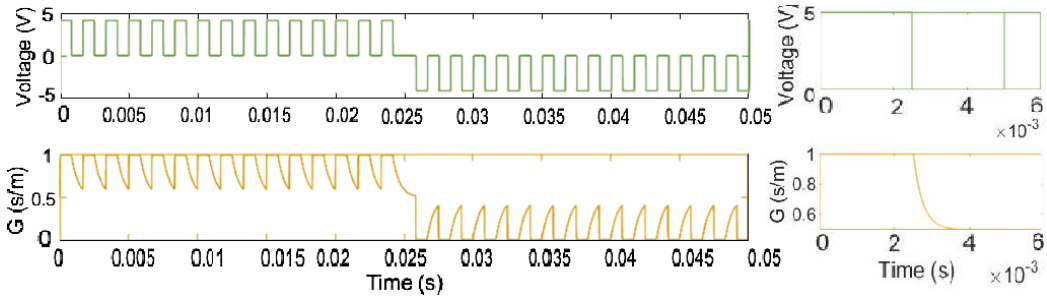


Figure 5.5: The conductance results of the dynamic memristor vary with the external voltage.

$$\begin{cases} sign^{(x)} = 1 \quad \text{if} \quad x > 0 \\ sign^{(x)} = 0 \quad \text{if} \quad x \leq 0. \end{cases} \tag{5.32}$$

Several variables I, V, and G are involved in the model, which respectively represent the output current, input voltage, and conductance. The parameters r and a, whose values are included in the Table 5.1, have been chosen based on the physical material properties required for fitting the model. Furthermore, the parameters K and G_{th} are varied with respect to V, while K_n and K_p are parameters used to match the characteristics of the dynamic memristor model, and their corresponding values also provided in the Table 5.1. $sign$ represents the sign function.

The V-I characteristic curves of the dynamic memristor model is shown in Figure 5.4. The data obtained from the voltage scan is depicted by the solid line, while the dotted line represents the data generated by simulating the mathematical model of the memristor. The

direction of the voltage scan is indicated by the arrows. The output current of the dynamic memristor will be sampled and used as the reservoir states.

Figure 5.5 depicts the conductance (memristor state) variation of the dynamic memristor. Its conductance nonlinearly decreases when the voltage switches from 5V to 0V in the positive direction, and increases when the voltage switches from -5V to 0V in the negative direction. When the voltage keeps 0, the conductance will decrease, which is regarded as the fading memory. As illustrated in Figures 5.4 and 5.5, the nonlinearity and fading memory characteristics indicate the suitability of using the dynamic memristor as the reservoir in our work. Specifically, we will use the current of the memristor as the reservoir states.

Nonvolatile Memristor Model

In this work, a NonVolatile Memristor model (NVM) will be used to construct the Memristor-based Window Element (MWE) and Memristor-based Delay Element (MDE), which will be used as memristive elements to construct the fully analog memrsitive TDR system. This NVM memristor model is as follows:

$$V^{(t)} = (R_{on}\frac{x^{(t)}}{D} + R_{off}(1 - \frac{x^{(t)}}{D}))i^{(t)}, \tag{5.33}$$

$$\frac{dx^{(t)}}{dt} = \frac{\mu_v R_{on}}{D} i^{(t)}, \tag{5.34}$$

$$\begin{cases} \mu_v &= \mu_0 \quad \text{if} \quad \left|V^{(t)}\right| \geq V_{th} \\ \mu_v &= 0 \quad \text{if} \quad \left|V^{(t)}\right| < V_{th}, \end{cases} \tag{5.35}$$

where the minimum and maximum memristance are represented by R_{on} and R_{off}, respectively. The current flowing through the memristor and the voltage across it are denoted by $i^{(t)}$ and $V^{(t)}$, respectively. D, μ_v, and $x^{(t)}$ refer to the effective length, dopant mobility rate, and length of the doped region of the memristor, respectively. Moreover, under different ap-

Table 5.1: The parameter setting of two memristor models

The dynamic Memristor	Value	The HP Memristor	Value
G_0	0.5	V_{th}	$\pm$1V
r	0.99	R_{on}	100Ω
α	0.23	R_{off}	40kΩ
K_p	9.13	μ_0	10^{-16}
K_n	0.32	D	$10nm$

plied voltages, the value of μ_v varies according to a threshold parameter V_{th}. Its parameters are listed in Table 5.1.

5.3.2 Dynamic Input Masking Module (DIM)

Figure 5.6 shows the circuit diagram of dynamic input masking module. Following Section 5.2, the input signal within the sliding window will be accumulated and averaged, and then it will be held and added to the current input signal by the weights, where the dynamic mask layer shown in Figure 5.6 can implement the function presented in Equations (5.14) and (5.16). According to Kirchhoff's law, the output voltage of the OPE is:

$$V_F = -\sum_{i=1}^{M} \frac{R_f}{R_s} V_{Ii}. \tag{5.36}$$

where R_f and R_s are resistances as depicted in Figure 5.6 and V_{Ii} is the input from the i-th row and M is the size of memristive array's row. The output voltage of the j-th column of memristive crossbar V_{Oj} is:

$$V_{Oj} = -\left[\sum_{i=1}^{M}(R_0 \times G_{ij} \times V_{Ii}) + \frac{R_0}{R_f} V_F\right], \tag{5.37}$$

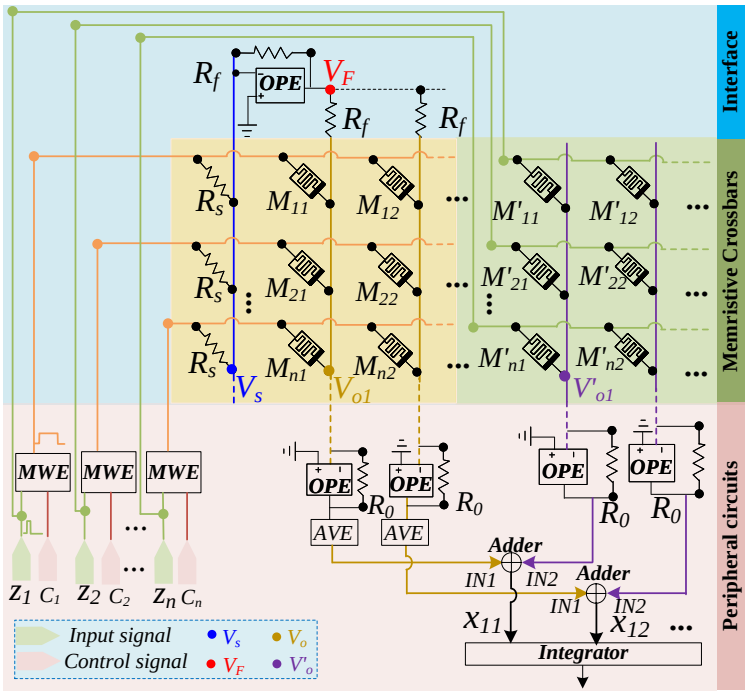


Figure 5.6: Circuit schematic of memristor-based dynamic input mask layer. V_s denotes the voltage at the inverting input terminal of OPE, and V_F denotes the output voltage of the OPE. $IN1$ and $IN2$ represent two components of the dynamic masking proposed in Equation (5.14).

where G_{ij} represents the conductance of the memristor in i-th row and j-th column. Combining Equation (5.36) with Equation (5.37), V_{Oj} can be described as:

$$V_{Oj} = \sum_{i=1}^{M} R_0 \times (G_s - G_{ij}) \times V_{Ii}. \tag{5.38}$$

where G_s represents the conductance of the resistor R_s. When $R_0 = R_s$, V_{Oj} can be rewritten as:

$$V_{Oj} = \sum_{i=1}^{M} (1 - R_0 \times G_{ij}) \times V_{Ii}. \tag{5.39}$$

After the average operation to V_{Oj}, the form of Equation (5.39) is in accordance with the second term of Equation (5.15), forming our input signal $IN1$. Then another part of input signal is $IN2$ (the first term of Equation (5.15)), which is $V'_{Oj} = \frac{R_0}{R_{ij}} \cdot V_{Ii}$, will be added

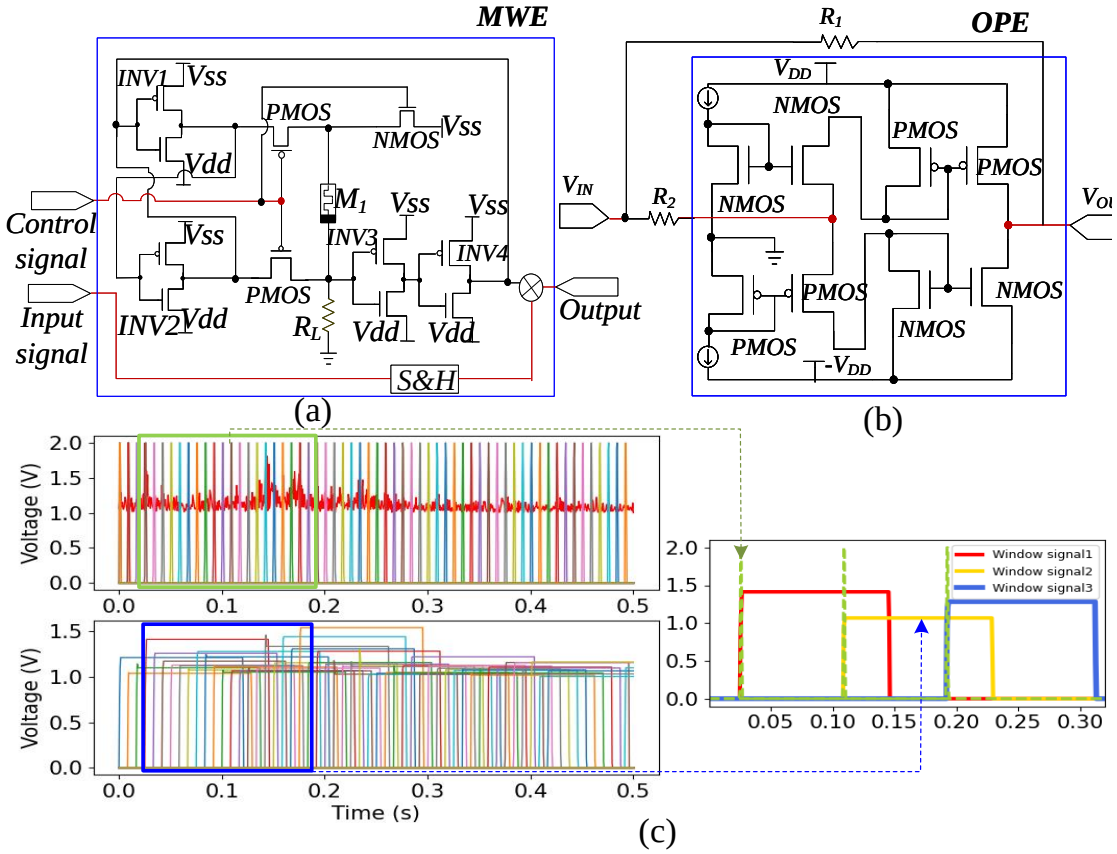


Figure 5.7: (a) Circuit schematic of memristor-based window element (MWE). (b) Circuit schematic of OPE. It applies the conduction characteristics of PMOSs and NMOSs, which draws on the current conveyor. (c) Circuit simulation results of the memristor-based window element.

with $IN1$, where R_{ij} is the memristance value of the memristor. Through this, the input will be dynamically masked by our proposed memristor-based crossbar architecture.

Regarding the circuit schematic diagram of our proposed memristor-based dynamic input mask layer shown in Figure 5.6, there are several sub-circuit units fabricated by the black boxes, such as OPE and Memristor-based Window Element (MWE). The OPE shown in Figure 5.7 (b) draws on the design of the current conveyor [250], which realizes a function similar to the operational amplifier.

Moreover, a MWE is proposed to generate the masked input signal masked by an adaptive window, where the pulse width of this input signal could be evolved adaptively

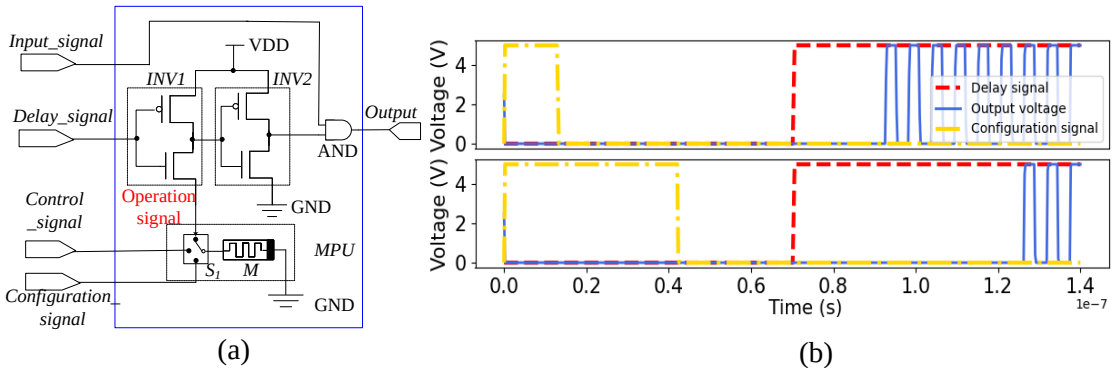


Figure 5.8: (a) Circuit schematic of the memristor-based window element. (b) Circuit simulation results of the memristor-based window element.

to various tasks. Figure 5.7 (a) shows this MWE. When $control\_signal = 0$, two PMOSs are turned on, NMOS is turned off, and the output signal Q is connected to the input terminal of the inverter $INV1$, so the voltage across the memristor will quickly switch its resistance. When $control\_signal = 1$, PMOS is turned on, and two NMOSs are turned off. The resistance value of memristor M_1 remains unchanged because the applied voltage V_{SS} is lower than its threshold voltage. The voltage divider between the value resistors R_L remains unchanged, so the state of the output signal Q remains unchanged. When $control\_signal$ changes from a low level (0) to a high level (1), which means that the rising edge of the pulse arrives, since the memristance of M_1 has been switched, the division voltage between the memristor M_1 and resistor R_L will also change. After the inverters, $INV3$ and $INV4$, the state of the output signal will be inverted.

The circuit simulation results of the proposed memristor-based window element are displayed in Figure 5.7 (c). The upper one presents the raw input data (red points) and the sampling signals (short pulse). The bottom one presents the masked input to the sampled signals. The right one shows the enlarged parts that involved the sampling signals (green pulse) and masked signals. Under the $control\_signal$ with different pulse widths, there will be different window sizes of the masked input.

5.3.3 Memristor-based Delay Element (MDE)

A single dynamic memristor is utilized to build a nonlinear node with nonlinear dynamic behavior and fading memory. In order to implement the technique of state forwarding, we require memristors with nonvolatile memory rather than volatile memristors with fading memory since the delay volume will not be changed under one TDR setting. Memristor-based delay element (MDE) [264] is applied to implement the multi-dependency delay of memristive TDR. According to Equation (5.13), the conventional TDR model incorporates only one dependency that forwards from the last state $x^{(t-1)}$, which might restrict its ability to make accurate predictions for long-term memory datasets. To increase its memory capacity, the matching delay element is necessary. The MDE is demonstrated in Figure 5.8 (a). It is made up of two inverters, $INV1$ and $INV2$, as well as a multiplier and a Memristive Programming Unit (MPU).

In this work, we utilized the memristive reconfigurable unit proposed by Yang et al. [338] as the MPU in the delay element. This MPU consists of one memristor and four transistors, and its memristance R_m can be adjusted using the $configuration\_signal$ and $control\_signal$. The MDE operates in two phases, namely configuration and operation. During the operation phase, when the $control\_signal$ is positive, S1 connects to the operation signal and the delay element functions. On the other hand, during the configuration phase, when the $control\_signal$ is negative, S1 connects to the $configuration\_signal$, and the delay element operates. Varying the length of the $configuration\_signal$ leads to different memristance values. The delay duration of changing from low to high can be expressed using the Elmore delay model [135] as:

$$t_{pLH} = 0.69(R_{eqn1} + R_{eqn2} + R_{eqn3} + R_m)C_L, \tag{5.40}$$

where R_m represents the memristor value of M. We can apply the following equation [327]

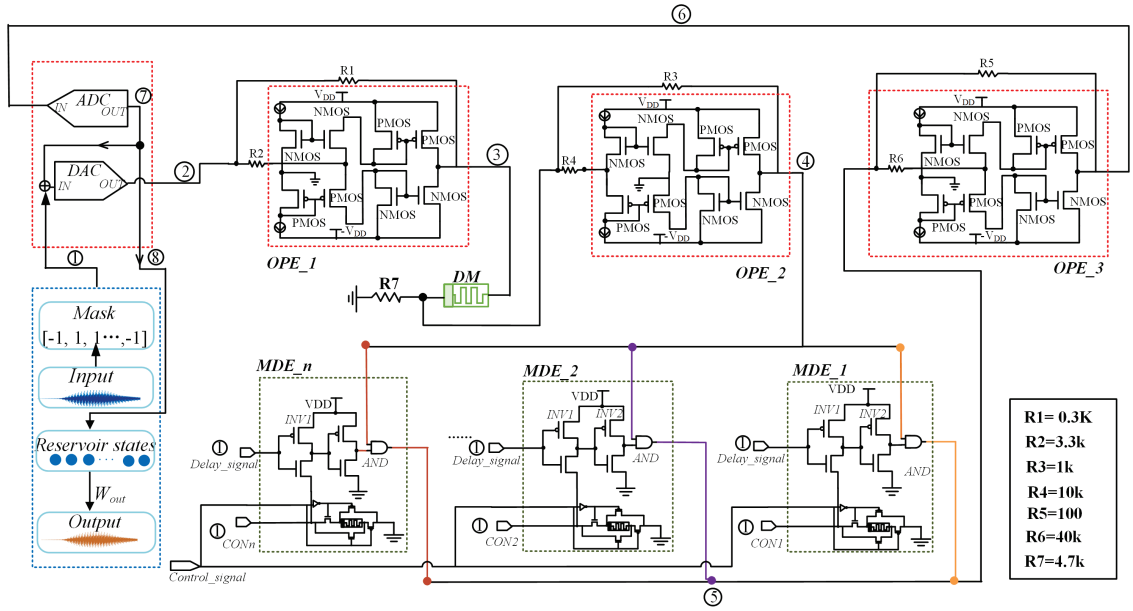


Figure 5.9: Memristive implementation of adaptive memory-enhanced TDR.

to calculate the equivalent on-resistance R_{eqn1}, R_{eqn2} and R_{eqn3}:

$$\begin{aligned} R_{eq} &= -0.5V_{DD}\int_{V_{DD}}^{V_{DD}/2} VdV/I_{DSAT}(1+\lambda V) \\ &\approx 3V_{DD}/4I_{DSAT}(1-7\lambda/9). \end{aligned} \tag{5.41}$$

Here, V_{DD} denotes the supply voltage, I_{DSAT} refers to the transistor's current in the saturation region, and λ presents the modulation factor of channel length. The circuit simulation result of MDE is shown in Figure 5.8 (b).

5.3.4 Architecture of Memristive Memory-enhanced TDR Through Reservoir States

The architecture diagram of the memristive implementation of memory-enhanced TDR is shown in Figure 5.10. The proposed memory-enhanced TDR can be implemented by a dynamic memristor and memristor-based delay elements. Moreover, personal computer (PC)

and necessary peripheral circuits are also needed for the circuit experiments. The PC is used to run the basic loops of proposed algorithm, and the peripheral circuits are required to interact between the algorithm and the memrisitve memory-enhanced reservoir, where OPE represents the proportional operation module to implement the function of proportion. In this work, we implement this architecture by circuit simulation on NGSPICE and interaction with algorithm loop on Python. For sake of the convenient circuit simulation, we applied the circuit shown in Figure 5.7 as one form of OPE that draws on the design of the current conveyor, where the relationship between the input voltage and the output voltage is $V_{OUT} = (1 - \frac{R_1}{R_2})V_{IN}$.

The circuit schematic diagram of the memristive implementation is shown in Figure 5.9. Executing the memristive memory-enhanced TDR consists of 8 steps, related steps also have been marked in the Figure 5.9:

- **Step 1:** The first step is about the input data pre-processing and MDEs configuration. As for input data pre-processing, the input data will be discretized and normalized the time series between -1 and 1 by mask. As for the MDE configuration, the MDEs will work in configuration phases to program the memristance of M_1 to M_n.

- **Step 2:** Added with the output signal of ADC, DAC module will generates the voltage pulse with the amplitude (0-3.3V) and pulse width of $120\mu s$ corresponding the summed data value of the ADC output and input data.

- **Step 3:** The amplitude of the generated pulse will be changed to the range of -3 to 3V by OPE_1, and applying to the dynamic memristor DM.

- **Step 4:** The current of DM, I_{DM}, will be transformed into voltage by a constant resistor R_7, of which value is $I_{DM} \times R_7$. And OPE_2 is use to amplify the amplitude of the voltage to a larger range (0-3.3V).

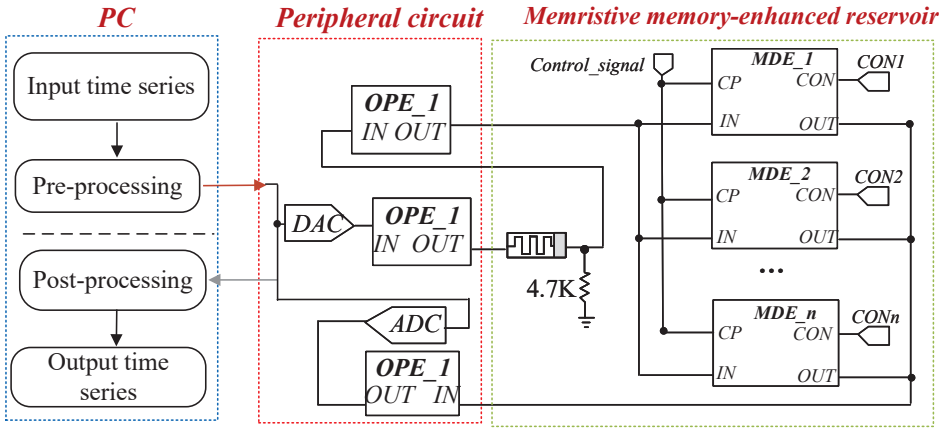


Figure 5.10: The memristive implementation framework of proposed adaptive memory-enhanced TDR.

- **Step 5:** The output signal of OPE_2 represent the current state of DM in the form of voltage. Then, this signal will be sent to MDEs in parallel.

- **Step 6:** OPE_3 will integrate the signals from all the MDE, giving a factor to all the delay signals.

- **Step 7:** ADC module will sample the integrated signal from OPE_3, which will be further transformed to the input terminal of DAC for the next loop of training (back to *Step 1*) or to the PC for the post-processing (moving to *Step 8*). We implement the reservoir part of the proposed model into the memristive circuits, thus the DACs and ADCs are required for transferring the signals between memristive reservoir circuit and input/output layers.

- **Step 8:** The output of the ADC module will be transformed to PC for the post-processing, which will be regarded as reservoir states and to calculate the output time series.

5.3.5 Architecture of Fully Analog Memristive TDR with Dual Memory Enhancement

Section 5.3.4 proposed the memristive implementation of the reservoir part (green box in Figure 5.11). Our current work implements all the components of our proposed work in Section 5.2 into the fully analog memristive architecture for TDR, which are dynamic input layer, reservoir layer and the output layer. Figure 5.11 shows the overall architecture of our proposed fully analog memristive architecture of our proposed TDR with enhanced memory. The pink and blue boxes represent the differential pair of dynamic masking layers, where the pink box implements the right part of the Equation (5.14), constructed the circuit based on DIM introduced in Section 5.3.2. The blue box implements the left part of the Equation (5.14), constructing the circuit based on MDE introduced in Section 5.3.3. The green box represents the reservoir layer of our proposed architecture, and the output layer is given in the grey box.

There are 8 steps to execute the memristive memory-enhanced TDR:

- **Step 1**: The first step is about the circuit initialization. We will first configure the memristor value of the nonvolatile memristors (HP memristor) in the dynamic masking layer, the reservoir layer, and the output layer, respectively.
- **Step 2**: The input signals Z will be fed into the memristive dynamic masking layer. By the operations of MWEs, the input signal Z will be sampled and held with an adaptive window period. After that, the signals will be fed into the differential pair of memristor-based crossbars (pink and blue boxes), then the sampled signals will be further weighted by the memristor-based crossbars with sparse memristance distribution.
- **Step 3**: After integrating the masked signals from the dynamic masking layer and changing them into -3V to 3V, they will be fed into the reservoir layer (green box).

The current of the dynamic memristor, I_{DM}, will be converted into voltage V_i through a load resistor R_L, whose value is $I_{DM} \times R_L$.

- **Step 4**: The signals will pass through the MDMs, where the reservoir states will be forwarded under the different delay signals. In MDMs, OPE_1 is used to amplify the amplitude of the voltage to a larger range (0-3.3V).
- **Step 5**: The current state of DM in the form of voltage is represented by the output signal of OPE_1. This signal is then transmitted in parallel to the MDEs.
- **Step 6**: OPE_2 is utilized to combine the signals from all MDEs, providing a weighting factor to each delay signal.
- **Step 7**: The voltage V_i from the dynamic memristor will be transformed to the output layer.

5.4 Experimental Studies of Memristive Memory-enhanced TDR Through Reservoir States

In Section 5.3.4, we introduce an adaptive memory-enhanced TDR. Specifically, we design a higher-order delay unit, which is capable of accumulating and transferring the long-history states in an adaptive manner to further enhance the reservoir memory. Particle Swarm Optimisation is applied to optimize the enhanced degree of memory adaptivity. This section introduces the experiments, where our experiments demonstrate its superiority both for short- and long-term memory datasets over seven existing approaches. In light of the hardware-friendly feature of TDR, we further propose a memristive implementation of our adaptive memory-enhanced TDR, where a dynamic memristor and the memristor-based delay element are applied to construct the reservoir. Through circuit simulation, the feasibility

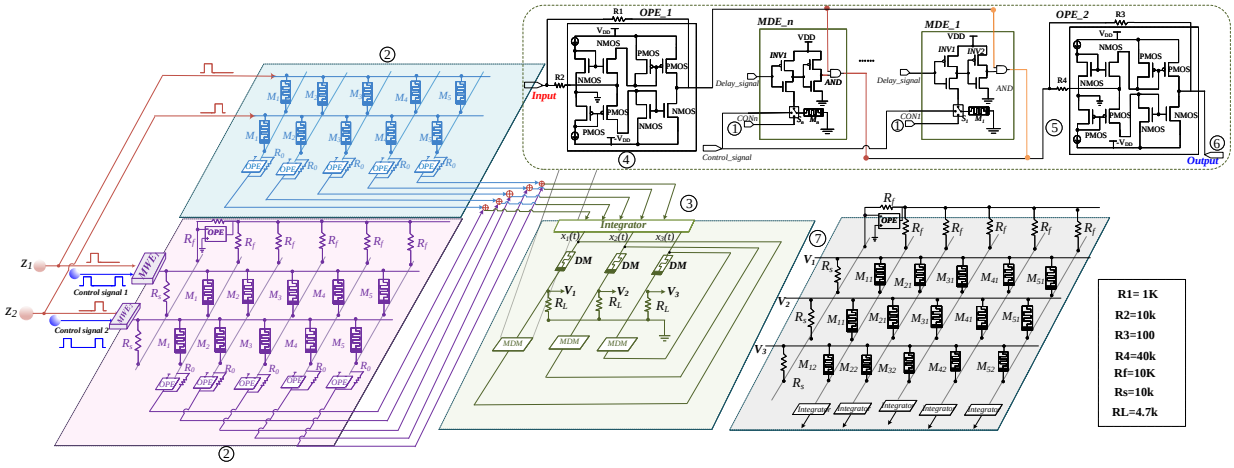


Figure 5.11: The circuit schematic of the full analog memristive Time Delay Reservoir (TDR) with enhanced memory features. The Dynamic Input Masking Module (DIM), depicted in pink and blue and labeled as '2', applies a dynamic mask to input signals using a memristor-based crossbar architecture, as detailed in Section 5.3.2 (referencing Figure 5.6). This module, integrating with the reservoir layer shown in green and labeled as '3', consists of nonlinear nodes and delay elements that implement state forwarding. The nonlinear nodes, dynamic memristors exhibiting nonlinear behavior and fading memory, are discussed in Section 5.3.1. In contrast, the delay elements, nonvolatile memristors that store and forward past states, are elaborated upon in Section 5.3.3 (also shown in Figure 5.8). Finally, the output layer, represented in grey and labeled as '7', is a linear readout layer using nonvolatile memristors for storing output weights, with its functionality and design principles detailed in Section 5.2.3.

of our proposed memristive implementation is verified. The comparisons with different hardware reservoirs show that our proposed memristive implementation is effective both for short- and long-term memory datasets, while exhibiting benefits in terms of smaller circuit area and lower power consumption compared with traditional hardware reservoirs.

5.4.1 Experimental Setup

Several works have discussed the memory term of different datasets, which could be divided into the long-term memory and short-term memory datasets according to the autocorrelation plots [354][90]. We visualize the autocorrelation plots of one short-term memory dataset (Narma10) and one long-term memory dataset (Nonlinear audio) in Figure 5.12.

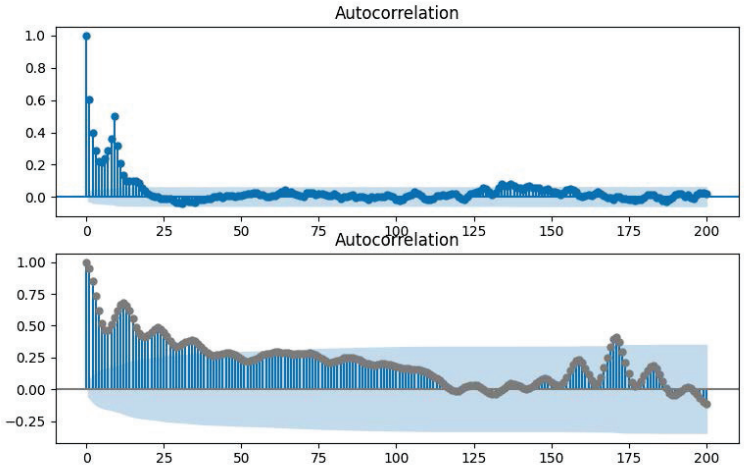


Figure 5.12: Autocorrelation plot of Narma10 (top) and Nonlinear audio dataset (bottom).

Long-term Memory Datasets

We use three real and one synthetic long-term memory dataset:

- **Nonlinear Audio:** searchers found that long memory appears to be strongly represented in music [90]. This dataset is from [115], which is a short recording of a Jazz quartet. The length of training, validation and test sets are set as 2000.

- **Tree Ring:** this dataset contains 4351 tree ring measures of a pine from Indian Garden, Nevada Gt Basin obtained from R package tsdl [2], where 2500 items are used for training, 1000 for validation and 850 for testing.

- **Dow Jones Industrial Average (DJI):** The raw dataset contains DJI daily closing prices from 2000 to 2019 obtained from Yahoo Finance [3], where 2500 items are used for training, 1500 for validation and 1029 for testing.

- **ARFIMA series:** We generated a series of length 4001 using the following model

[2] https://pkg.yangzhuoranyang.com/tsdl/
[3] https://finance.yahoo.com/

with obvious long memory effect [89]:

$$(1 - 0.7B + 0.4B^2)(1 - B)^{0.4} Y_t = (1 - 0.2B)\varepsilon_t \tag{5.42}$$

Where the length of training, validation and testing set are 2000, 1200 and 800, respectively.

Short-term Memory Datasets

We use two real and three synthetic short-term datasets:

- **Santa Fe Laser Set-A and Set-D:** Santa Fe Laser data set was used [4]. This is a section of a real laser that shows how its intensity oscillates between regular and chaotic patterns. This task is to predict the next value of the input sequence. Two different Santa Fe datasets were used, the first of which is the univariate time series A derived from laser-generated data, and the second is the computer-generated time series D. For both time series A and D, we discarded the first 200 items as washout, then used the next 2000 items for training, the next 4000 for validation, and the final 1800 for testing.

- **Narma10 and Narma20:** NARMA systems of order 10 and 20 are applied as short-term memory datasets, of which equations are [10]:

$$\begin{aligned} y\left(t+1\right) =& 0.3y\left(t\right) + 0.05y\left(t\right)\sum_{i=0}^{9} y\left(t-i\right) \\ &+ 1.5s\left(t-9\right)s\left(t\right) + 0.1, \end{aligned} \tag{5.43}$$

[4] http://web.cecs.pdx.edu/~mcnames/DataSets/index.html

Table 5.2: Parameter setting of SOTA models and our proposed work

Models	Parameter setting
RNN	hidden size=200; nonlinearity='tanh'
LSTM	hidden size=200;
mRNN	hidden size=200; nonlinearity='tanh'; K=100
mLSTM	hidden size=200; K=25
ESN	hidden size=200; nonlinearity='tanh'; leaking_rate=1,spectral_radius=0.9
Deep-esn	hidden size=100; num_layer=4; nonlinearity='tanh'; leaking_rate=1,spectral_radius=0.9
TDR	num_node=200; $\alpha = 0.6$; $\beta = 0.75$; $\theta = 0.2$
Our work	num_node=200; num_order=20; $\theta = 0.2$; population=20; max_iteration=200

$$y\left(t+1\right)=\tanh\left(0.3y\left(t\right)+0.05y\left(t\right)\sum_{i=0}^{19}y\left(t-i\right)+1.5s\left(t-19\right)s\left(t\right)+0.1\right). \tag{5.44}$$

We selected the NARMA sequences with 8000 items, where the first 200 items were discarded as washout, the following 2000 items were used as the training set, the following 4000 as the validation set, and the remaining as the testing set.

- **Hénon Map:** Hénon map has been established as a typical discrete-time dynamic system with chaotic behavior. It describes a nonlinear 2-D mapping that transforms a point $(x(n), y(n))$ on the plane into a new point $(x(n+1), y(n+1))$, which is [22]:

$$x(n+1)=y(n)-1.4x(n)^2, \tag{5.45}$$

$$y(n+1)=0.3x(n)+w(n). \tag{5.46}$$

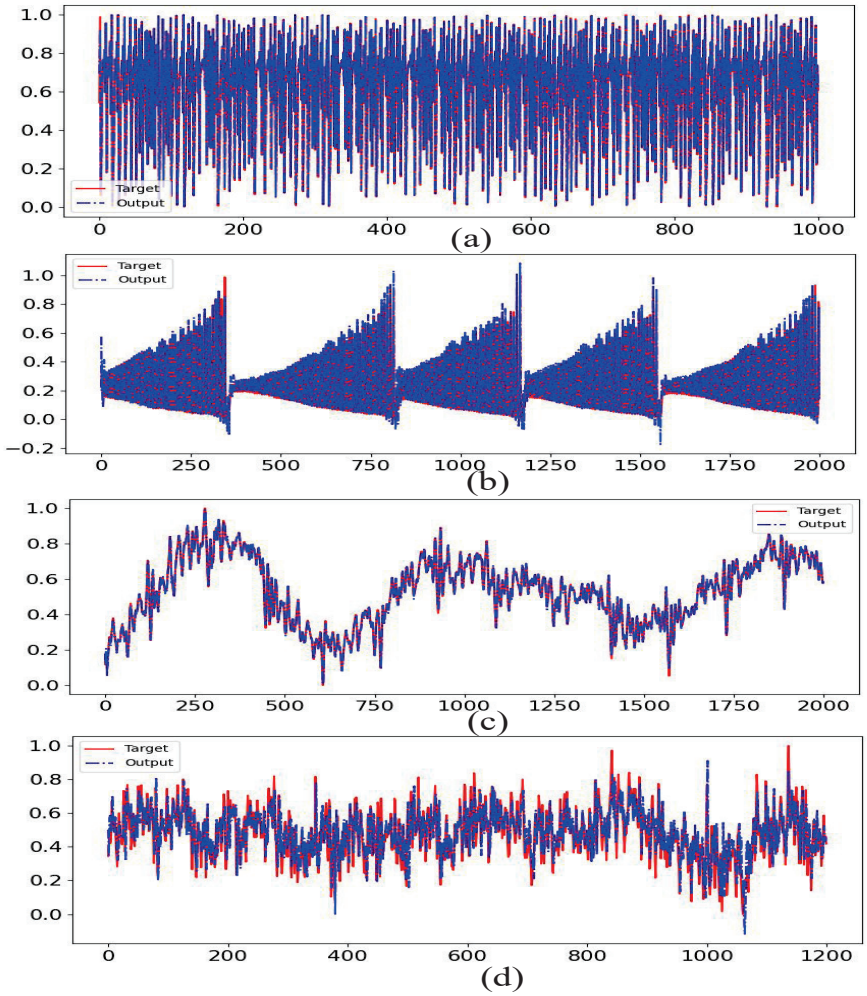


Figure 5.13: Actual predicted outputs of our proposed adaptive memory-enhanced TDR vs corresponding targets. (a) Hénon Map; (b) Santa set-A; (c) Nonlinear audio; (d) ARFIMA series.

5.4.2 Comparison With Different Software Models

With the objective of evaluating the predictive performance of the proposed memory-enhanced TDR, we conduct comparisons with several existing models for time series prediction, namely vanilla ESN [128], Deep ESN [182], conventional TDR [8], vanilla LSTM [241], vanilla RNN [85], a variant of LSMT (mLSTM) and a variant of RNN (mRNN) [354].

To compare our work with other state-of-the-art works, we use RMSE and average rank. RMSE measures the absolute error of each model, while average rank measures the

relative error of each model. The average rank is computed by ranking each model according to its RMSE value, where the model with the lowest RMSE gets the highest rank. Then, the average rank is obtained by averaging the ranks of each model across all the test cases. The better the average rank, the better the overall performance of the model. Root Mean Squared Error (RMSE) is used as a measure of predictive performance in our experiments:

$$RMSE = \sqrt{\left\langle \|\hat{y}(t) - y(t)\|^2 \right\rangle}, \tag{5.47}$$

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

The parameter settings of all approaches used in the experiments are listed in Table 5.2. For a fair comparison, the hidden size or the number of nodes of the different models are all set as 200. As for the mRNN and mLSTM, the hyperparameter K are set as 100 and 25, respectively, as existing work [354] has shown that larger K will lead to better performance for mRNN, while smaller K will be beneficial to mLSTM. As for Deep ESN, the parameters are set to the same values used in [182]. As for the PSO optimization part of our proposed method, the population is set as 20 and the maximum iteration is 200. Once the hyperparameters were set, the experiments were run 20 times for each dataset.

Figure 5.13 shows the superposition between the actual outputs of our proposed model and the corresponding targets for both of the short-term (Hénon Map and Santa Set-A) and long-term (Nonlinear audio and ARFIMA) memory datasets. They show that the signal generated by our proposed model mimics the desired signal. In order to further verify and evaluate our proposed model, the predictive performance with other existing models are given in Table 5.3, which lists the average RMSE for 20 runs of each model, and its average ranking (AveRan.) of one model on two types of dataset, as well the overall average ranking for all the datasets.

For the short-term memory datasets, mRNN can improve RNN by introducing a memory filter, while mLSTM and LSTM perform similarly on short-term memory datasets. Our proposed method can outperform other existing models on the short-term memory datasets (the average ranking is 1). Improvements in RMSE were obtained for all short-memory datasets and were particularly large for Hénon Map, where the improvements were from 0.5713 (RNN) to 0.0042 (memory-enhanced TDR). Improvements in the predictive performance on short-memory datasets probably occurred because the adaptive connection between the current states and the states in the short-term memory will be optimized by PSO. This is likely beneficial no matter whether the datasets are short-term or long-term memory datasets.

In terms of the average performance on long-term memory datasets, the performance of mRNN and mLSTM is better than RNN and LSTM, and Deep ESN performs better on long-term memory datasets than the short-term datasets did. Our proposed method outperformed the existing methods on the long-term memory datasets, obtaining average ranking of 1. Improvements were obtained in all long-term memory datasets and were particularly high on the Arfima task, where improvements were from 1.1620 (RNN) to 0.0866 (memory-enhanced TDR). The improvements obtained for the long-term memory datasets were greater than the improvements on the short-term memory datasets. The reason for that is twofold. First, the higher-order delay units can accumulate the states existing in the long-term memory and transfer them into the current state. Second, PSO optimization of the enhanced degree makes such “accumulation" to be established in a correct and adaptive way.

From the perspective of overall performance on all the datasets, the average ranking of our proposed method outperforms all other existing models. In summary, the memory of TDR can be enhanced by our proposed adaptive design. The proposed adaptive memory-enhanced TDR can outperform other exiting models on both short-term and long-term

memory datasets, where its performance improvement on long-term memory datasets is more obvious than that on short-term memory datasets. The Mann–Whitney U tests of the existing models with our proposed method are conducted, of which P value are given in Table 5.4. The level of significance is 0.05, therefore, we can confirm that our proposed adaptive memory-enhanced TDR can improve the predictive performance compared with the existing models. Overall, the proposed adaptive memory-enhanced TDR was verified on both short memory and long memory datasets. It improved not only improve prediction performance over the existing reservoirs such as ESN [128] and conventional TDR [8], but also over other heavier approaches such as RNNs [85], LSTMs [241], deep-ESN [182], and variants of RNN and LSTM [354].

Moreover, we also compare the prediction performance of our proposed memristor-based memory-enhanced TDR with our previous two works proposed in Sections 4.2.1 and 4.2.2 [260, 261], which use the pure-memristor RC model [286] instead of TDR model [8]. According to Table 5.3, the predictive performance of proposed memristor-based memory-enhanced TDR outperforms our previous works in terms of the average predictive performance [260, 261] over both of the long-term memory dependency and short-term memory dependency dataset. Benefiting by the enhanced memory, our proposed approach has better memory to process both of the long-term memory dependency and short-term memory dependency dataset. However, our newly proposed work requires higher circuit overhead than our previous two approaches [260, 261], which will be discussed in detail in Section 5.4.3.

5.4.3 Comparison With Different Hardware Reservoirs

We also compare our proposed memristive TDR with other hardware reservoirs, where the comparisons contain predictive performance comparison (shown in Table 5.3), and hardware performance comparison (shown in Table 5.5).

Table 5.3: Prediction performance comparison of different models

Datasets	Nar.10	Nar.20	SantaA	SantaD	Hénon Map	AvRan.*	Non. audio	Tree	DJI	Arfima	AvRan.	AvRan.
Type	Short-memory						Long-memory					S&L
Software model												
RNN [85]	0.0448	0.0667	0.0790	0.0398	0.5713	0.1603(5)	0.0277	0.2871	0.2605	1.1620	0.4343(8)	0.2821(8)
LSTM [241]	0.0415	0.0506	0.0536	0.0676	0.2843	0.0995(3)	0.0393	0.2833	0.2492	1.1340	0.4265(7)	0.2448(7)
mRNN [354]	0.0219	0.0584	0.0277	0.0463	0.2357	0.0780(2)	0.0543	0.2818	0.2487	1.0880	0.4182(5)	0.2292(2)
mLSTM [354]	0.0506	0.0571	0.0353	0.0532	0.3746	0.1142(3)	0.0231	0.2859	0.2531	1.1490	0.4278(6)	0.2535(5)
ESN [128]	0.0733	0.0731	0.0553	0.0612	0.0365	0.0599(6)	0.0327	0.1357	0.2192	0.1077	0.1238(4)	0.0883(4)
Deep-esn [182]	0.0986	0.0937	0.1186	0.0753	0.2752	0.1323(8)	0.0565	0.1290	0.1165	0.0954	0.0993(2)	0.1176(6)
TDR [8]	0.0577	0.0693	0.0427	0.0785	0.0653	0.0627(6)	0.0296	0.1386	0.1291	0.1070	0.1011(2)	0.0797(3)
Ours	**0.0186**	**0.0223**	**0.0246**	**0.0294**	**0.0042**	**0.0198(1)**	**0.0070**	**0.1158**	**0.1041**	**0.0866**	**0.0784(1)**	**0.0458(1)**
Hardware reservoir												
Zhong's [357]	0.0980	0.0907	0.0741	0.0463	0.0192	0.0656(3)	0.0117	0.2762	0.2391	1.150	0.4192(4)	0.2228(4)
Bai's [16]	0.0683	-	-	-	-	-	-	-	-	-	-	-
Proposed work in Section 4.2.1 [260]*	**0.0301**	0.0476	0.0593	0.0672	0.0488	**0.0506(2)**	0.0452	0.0739	0.0754	0.0885	**0.08185(2)**	**0.0645(2)**
Proposed work in Section 4.2.2 [261]*	0.0862	0.0885	0.0892	0.0928	0.0743	0.0862(4)	0.1370	0.1831	0.1955	0.2096	0.1813(3)	0.1285(3)
Ours	**0.0372**	**0.0463**	**0.0170**	**0.0237**	**0.0046**	**0.0257(1)**	**0.0097**	**0.1212**	**0.1096**	**0.0860**	**0.0816(1)**	**0.0505(1)**

* AvRan. 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.

* In order to ensure fair comparisons between our proposed works in Sections 4.2.1 and 4.2.2 (works [260, 261]) and our proposed TDR, the number of their reservoir states are set as the same.

Table 5.4: The results of Mann–Whitney U tests of SOTA models and our proposed model

Method	RNN	LSTM	mRNN	mLSTM	Work proposed in Section 4.2.1 [260]
P-value	0.0170	0.0136	0.0318	0.0211	0.0345
Method	ESN	Deep-ESN	TDR	Work proposed in Section 4.2.2 [261]	
P-value	0.0318	0.0067	0.0318	0.0012	

Table 5.5: Comparison of different hardware reservoirs

Methods	Implement.	Constructing a reservoir with same func. (no enhanced memory)				Constructing a memory-enhanced reservoir	
		#Mem*	#Mos	#Cap	Power	Realized?	Method
FPGA-based ESN [344]	FPGA	0	37n*	n	-	No	-
Mosfet crossbar-based ESN [158]	PCB	0	16n	0	-	No	-
CMOS-based LSM [238]	IC	0	24n	3n	-	No	-
BBD-based TDR [5]	PCB	0	128	1	-	No	-
CMOS-based TDR [16]	IC	0	57	3	$526\mu w$	No	-
Memristor-based TDR [205]	IC	1	0	0	$300\mu w$	No	-
Memristor-based TDR with mask [357]	IC	1	0	0	$50\mu w$	No	-
4T1R memristive reservoir [260]	IC	1	0	0	$0.058\mu w$	No	-
1R memristive reservoir [261]	IC	1	0	0	$0.027\mu w$	No	-
Memristor-based memory-enhanced TDR (Ours)	IC	1	0	0	$50\mu w$	Yes	Introducing MDEs (Power=$57.8\mu w$)

n represents the number of neurons existing in a reservoir. In order to construct a reservoir for ESN or LSM, n always starts from 5. The memory enhancement of our proposed method was removed in the columns corresponding to no enhanced memory.
#Mem, #Mos and #Cap refer to the number of memoristor, Mosfets and capacitors. Work haven't provide the power consumption marked as -.

From Table 5.3, we can see that our hardware implementation obtained better RMSE than other hardware-based reservoirs in all datasets where their performances were available for comparison. Moreover, the RMSE of the hardware implementation was competitive against the software implementations, sometimes even leading to better results than its software counterpart.

Table 5.5 summarizes the design specification of our proposed memristor-based memory-enhanced TDR and other state-of-the-art RC designs, where works [344] and [158] are related to the hardware implementations of ESN, work [238] is the hardware implementation of LSM, works [5] [16] [205] [357] focus on the hardware implementations of TDR. We compare the number of components used to construct a reservoir with the same function, where the "same function" refers to the basic features of the reservoir.

We can see that TDR is more hardware friendly compared with ESN models. We first

compare our proposed method against others by removing its enhanced memory. We can see that ESN models have to use n neurons to construct the reservoir, while TDR models can just use one nonlinear delay node to construct one reservoir. The relationship between the number of neurons (n in the figure) and the number of Mosfets (#Mos in the Fig) is shown in Figure 2.4. Taking FPGA-based ESN [344] as an example, 37 Mosfets and 1 capacitor are required to construct one neuron, and there may be more than 100 neurons used to construct a reservoir in their work, so that 3700 Mosfets will be existed in its circuit counterpart. However, TDR hardware implementations can prevent this overhead, where the number of Mosfets will remain unchanged with the increasing neurons. As for CMOS-based TDR [16], 57 Mosfets and 3 capacitors are required to construct a reservoir, which is more compact than the ESN reservoir hardware [344][158]. Moreover, as for the memristive TDR work, there will be memristors or memristor-based elements to construct reservoir instead of relying on the Mosfets. In terms of power consumption, the power consumption of CMOS-based TDR [16] is 526μw, where the nonlinear node requires most of the total power consumption. However, as for memristor-based TDR, there is only a dynamic memristor constructing the nonlinear node of TDR instead of CMOS circuits, so that the power consumption of memristor-based TDR [205] [357] is much lower than that of CMOS-based TDR. As we can see, the basic constructs of our proposed method are competitive, leading to the use of fewer and smaller components than non-memristive reservoirs and offering low power consumption. We also compare the hardware aspects of our proposed approach with our two previous approaches proposed in Chapter 4 in Table 5.5. The table shows that the two previous approaches use regular 4T1R or 1R memristor-based crossbars, which have very low energy consumption. Based on the power consumption calculation method for memristor-based crossobar [358], the power of 4T1R memristive reservoir could be calculated as $0.058\mu W$, and the power of 1R memristive reservoir could be calculated as $0.027\mu W$, which are large smaller than our newly proposed one. However, due to these concise and regular architecture, their predictive performance is limited as discussed in Section 5.4.2.

In addition, our proposed method is the only one that can adapt memory enhancement. To realize this enhanced memory of TDR, MDE was introduced, composed of 1 memristor, 8 Mosfets, 0 capacitor, and 2.98 μw power for one MDE. The average power of one MDE is computed as $2.89\mu w$ considering two working phases of MDE (configuration and operation). The number of MDEs applied for enhancing the memory of MTDR was 20. Therefore, there will be $2.89\mu w \times 20 = 57.8\mu w$ for enhancing the memory by higher-order delay unit. After introducing the higher-order delay unit for MTDR, the overall power is still less than that of a memristive TDR work proposed by Moon et al.[205]. The other two memristor-based TDRs [205] [357] only contain one memristor to construct their reservoirs, which is very circuit compact and energy-efficient. The power consumption of our memory-enhanced approach is higher than that of the non-memory-enhanced approach from [357]. However, the easiest reservoir architecture [357] also limits the memory ability for predicting long-term dependency tasks (see the hardware reservoir comparisons in Table 5.3), since the current reservoir states only come from the neighbouring dependency without the history and longer-term memory dependency. Our proposed memristive TDR expands the delay into higher order. For the sake of this expanded structure, as shown in Table 5.3, our proposed memristive memory-enhanced TDR outperforms other hardware implementations of reservoir on both of the short-term and long-term memory datasets. The gains in prediction performance of our approach against this approach are large, as shown in Table 5.5. Therefore, when opting for one of these approaches, there is a trade-off between prediction performance and power consumption.

5.5 Experimental Studies of Fully Analog Memristive Architecture for TDR with Dual Enhanced Memory

As Section 5.3.5 introduced, we have introduced a full analog memory-enhanced TDR. Specifically, two parts have been used to enhance the memory capacity of the reservoir. First, the interaction between the neighboring time steps is strengthened by dynamic masking, where a moving window with an evolvable window width is proposed to adapt to different tasks. Second, a reservoir state forwarding technique is applied to accumulate and jump adaptively between long-term states to further enhance the memory capacity of the reservoir. In this section, the feasibility of our proposed memristive implementation is verified. The comparisons between different hardware reservoirs and our proposed memristive implementation on short- and long-term memory datasets will be carried out.

The experimental setup will be given in Section 5.5.1. Ablation study will be introduced in Section 5.5.2. A comparison of predictive performance against existing software approaches will be introduced in Section 5.5.3. Comparisons with other existing hardware reservoirs will be given in Section 5.5.4.

5.5.1 Experimental Setup

Memory dependencies in datasets can be classified as long-term or short-term using autocorrelation plots [354, 90], which has been introudced in Section 5.4. The datasets applied to the experimental studies of fully analog memristive architecture for TDR with enhanced memory are the same with those applied in Section 5.4. Besides these prediction task, a classification task, which is a long-term dependency task, is also applied to very our proposed approach.

Long-term Memory Datasets

Four real and one synthetic long-term memory dataset are used to our experiments:

- **Nonlinear Audio:** long memory in music is strongly represented [90], and a recording of a Jazz quartet is used with training, validation, and test sets of length 2000 [115].
- **Tree Ring:** it contains 4351 measures from a pine in Indian Garden, obtained from R package tsdl [5]. 2500 items are used for training, 1000 for validation, and 850 for testing.
- **Dow Jones Industrial Average (DJI):** it consists of daily closing prices from 2000 to 2019 obtained from Yahoo Finance [6], where 2500 items are used for training, 1500 for validation and 1029 for testing.
- **ARFIMA series:** a time series data with long-term memory effect was generated using the following model [89]:

$$(1 - 0.7B + 0.4B^2)(1 - B)^{0.4}Y_t = (1 - 0.2B)\varepsilon_t, \tag{5.48}$$

  where B represents the backshift operator, Y_t is the time series at time t, and ε_t represents the error term. The generated dataset was divided into training (2000), validation (1200), and testing (800) sets.
- **Dynamic Gesture Recognition:** the dataset used in this study is a spatiotemporal time series data representing dynamic gestures, captured in 3D and recorded at a sample rate of 10 Hz [358]. The dataset was split into training (600 samples), validation (300 samples), and testing (300 samples) subsets.

[5] https://pkg.yangzhuoranyang.com/tsdl/
[6] https://finance.yahoo.com/

Short-term Memory Datasets

Three real and two synthetic short-term datasets are applied to our experiments:

- **Santa Fe Laser Set-A and Set-D:** the Santa Fe Laser dataset is used [7], which includes two time series, A and D. A is derived from laser-generated data, and D is a computer-generated time series exhibiting periodic to chaotic intensity pulsations. For both time series, we used the first 2000 items for training, the next 4000 for validation, and the final 2000 for testing.

  •**Narma10 and Narma20:** NARMA systems of order 10 and 20 are applied as short-term memory datasets [10]:

$$\begin{aligned} y(t+1) =& 0.3y(t) + 0.05y(t)\sum_{i=0}^{9} y(t-i) \\ &+ 1.5s(t-9)\,s(t) + 0.1, \end{aligned} \tag{5.49}$$

$$\begin{aligned} y(t+1) = \tanh(& 0.3y(t) + 0.05y(t)\sum_{i=0}^{19} y(t-i) \\ &+ 1.5s(t-19)\,s(t) + 0.1). \end{aligned} \tag{5.50}$$

  We used 2000 points as the training set, the following 4000 as the validation set, and 2000 as the testing set.

  •**Hénon Map:** The Hénon map is a well-known 2D nonlinear mapping that exhibits chaotic behavior [22]:

$$x(n+1) = y(n) - 1.4x(n)^2, \tag{5.51}$$

$$y(n+1) = 0.3x(n) + w(n). \tag{5.52}$$

  The data splitting setting is the same as Narma tasks.

[7] http://web.cecs.pdx.edu/~mcnames/DataSets/index.html

5.5.2 Ablation Study

In this study, we evaluated the impact of various components of our proposed algorithm on its performance. Table 5.7 presents a comparison of the results obtained when ablating the dynamic mask, state forwarding, and both components. Mann–Whitney U tests of the ablated approaches with all-equipped approach are conducted, and their p-values are also shown in Table 5.7. Our findings demonstrate that the removal of either the dynamic mask or state forwarding leads to a decrease in the performance of the proposed algorithm on both short-term and long-term datasets. These results suggest that both the dynamic mask and state forwarding are important components of the proposed algorithm and contribute significantly to its overall effectiveness.

The results in Table 5.7 suggest that the impact of removing the dynamic mask and state forwarding on the performance of the proposed algorithm varies depending on the type of dataset. Specifically, the removal of the dynamic mask appears to have a greater negative impact on the performance of short-term datasets compared to the removal of the state forwarding. On the other hand, the removal of the state forwarding appears to have a greater negative impact on the performance on long-term datasets compared to the removal of the dynamic mask. These findings suggest that dynamic masking is more effective at enhancing short-term memory, while state forwarding is more effective at enhancing long-term memory, corroborating our expectations according to the theoretical analysis presented in Section 5.2.5.

5.5.3 Comparison With Existing Software Models

We evaluated the effectiveness and efficiency of our memory-enhanced TDR model by comparing it with several state-of-the-art software models for time series prediction and clas-

Table 5.6: Parameter setting of the SOTA models and our propopsed model

Model	Parameter setting
RNN	hidden_size=200; nonlinear.='tanh'; $p = 0.89$; $g = 0.23$; $LR = 0.89$
LSTM	hidden size=200; $win\_size = 35$
mRNN	hidden_size=200; nonlinearity='tanh'; $p = 0.89$; $g = 0.23$; $LR = 0.89$ $K = 100$
mLSTM	hidden size=200; K=25; winsize=37
ESN	hidden_size=200; nonlinear.='tanh' IS=0.78; RS=0.69; LR=0.91; pho=0.87; C=0.52
Deep-ESN	hidden_size=200; nonlinearity='tanh' $IS = 0.78$; $RS = 0.69$; $LR = 0.91$; $\rho = 0.87$; $C = 0.52$
TDR	num_node=200; $\alpha = 0.6$; $\theta = 0.2$; $\beta = 0.75$;
Our work	num_node=200; $\theta = 0.2$; num_order=20; population=20; max_iteration=200; $c_1 = 0.5$; $c_2 = 0.5$; $w = 0.8$; $\sigma_k = 1 - e4$

Table 5.7: Experimental results with different algorithm components ablation

Different ablated components	Short-term dataset						Long-term dataset					
	Naram 10	Naram 20	SantaA	SantaD	Henon	Ave. Ran.	Audio	Tree	DJI	Arfima	DGR	Ave. Ran.
No dynamic mask (random mask)	0.0371 (3)	0.0463 (3)	0.0470 (3)	0.0331 (3)	0.0102 (3)	3	0.0097 (3)	0.1212 (2)	0.1096 (2)	0.0850 (2)	92.56% (2)	2
No state forwarding (basic state)	0.0240 (2)	0.0381 (2)	0.0240 (2)	0.0237 (2)	0.0092 (2)	2	0.0096 (2)	0.1243 (3)	0.1243 (3)	0.0866 (3)	90.64% (3)	3
Both ablated	0.0980 (4)	0.0907 (4)	0.0741 (4)	0.0463 (4)	0.0192 (4)	4	0.0117 (4)	0.2762 (4)	0.2391 (4)	0.1036 (4)	88% (4)	4
All equipped	**0.0103 (1)**	**0.0220 (1)**	**0.00013 (1)**	**0.0197 (1)**	**0.0058 (1)**	**1**	**0.0093 (1)**	**0.1106 (1)**	**0.0107 (1)**	**0.0761 (1)**	**96.34% (1)**	**1**
P-value	No dynamic masking VS All equipped: 0.00253; No state forwarding VS All equipped: 0.00253; Both ablated VS All equipped: 0.0483											

sification. These include the vanilla Echo State Network (ESN) [128], Deep ESN [182], conventional TDR [8], vanilla Long Short-Term Memory (LSTM) [241], vanilla Recurrent Neural Network (RNN) [85], a LSTM variant (mLSTM) [354], and a RNN variant (mRNN) [354], respectively. The mRNN model incorporates a memory filter to improve the RNN's performance. Furthermore, while mLSTM and LSTM yield similar results on short-term memory datasets.

We apply Root Mean Squared Error (RMSE) in our experiments as a performance metric for prediction, and is defined as [9]:

$$RMSE = \sqrt{\langle |\hat{y}(t) - y(t)|^2 \rangle}, \tag{5.53}$$

where $\hat{y}(t)$ and $y(t)$ represent the readout output and the desired output, respectively. The Euclidean norm is denoted by $|\cdot|$ and the empirical mean is represented by $\langle \cdot \rangle$. To ensure a fair comparison, all models were configured with a hidden layer size of 200 nodes,

including our proposed one. Moreover, we also apply PSO to other SOTA models to optimize their performance. Both our proposed method and the SOTA models use the same parameter values for PSO, which are set as $c_1 = 0.5$, $c_2 = 0.5$, $w = 0.8$, $population = 20$, $max\_iteration = 200$. The global parameters of the SOTA models will be optimized by PSO, such as the input scaling factor IS, reservoir scaling factor RS, leaky rate LR, spectral radius ρ, connectivity C for ESN. As for the vanilla RNN, there are also several parameters that will be optimized, including the sparseness of the connection p, the spectral scaling factor g, and the leaky rate LR. As for the vanilla LSTM, win_size will be optimized. Regarding the improved variants of RNN and LSTM, mRNN and mLSTM, there is one more memory parameter, k, which is needed to be optimized. The hyperparameters used with PSO for the experiments are listed in Table 5.6. The experiments were then conducted 20 times per dataset using these fixed hyperparameters.

According to the results presented in Table 5.8, our TDR approach with memory enhancements outperforms existing methods on short-term memory datasets, achieving an average ranking of 1. Our approach improves RMSE on all short-term memory datasets, especially on the SantaA dataset, where RMSE improved from 0.0973 (Deep-esn) to 0.00013 (ours). These results demonstrate the effectiveness of our approach in optimizing the interaction between neighboring time steps using PSO, leading to superior predictive performance on short-term memory datasets compared to existing methods. This enhanced interaction helps to increase the memory capacity of the TDR. These results are in accordance with the results shown in the ablation study in Table 5.7, where the performance is more degraded on the short-term datasets when the dynamic masking is ablated. We also compare the prediction performance of our proposed memristor-based dual memory-enhanced TDR with our previous three works [260, 261, 264], where two proposed works in Sections 4.2.1 and 4.2.2 ([260, 261]) use the pure-memristor RC model [286] (proposed in Sections 4.2.1 and 4.2.2), and one proposed work is the TDR model with the memory enhancement through the reser-

voir states (proposed in Section 5.3.4). Table 5.8 shows that our proposed memristor-based TDR with dual-memory enhancement achieves better predictive performance than our previous works [260, 261, 264] on most long-term and short-term memory dependency datasets. The advantage of our proposed approach over the other methods is that it implements the dual-memory enhancement, which enables it to process both the long-term and short-term memory dependency datasets more efficiently and accurately. The dual-memory enhancement consists of two components: the enhanced memory through dynamic mask and through reservoir states, which enables the first ranking of the prediction performance over other approaches. We performed Mann–Whitney U tests to compare our proposed method with the existing models, and reported the P values in Table 5.9. Using a significance level of 0.05, we confirmed that our proposed TDR with dual memory enhancement enhanced the predictive performance over the existing models.

In terms of performance on long-term memory datasets, mLSTM and mRNN outperformed LSTM and RNN on long-term memory datasets, while Deep ESN showed better performance on long-term memory tasks than short-term ones. Our proposed TDR approach with memory enhancements achieved the best average ranking and significantly improved predictive performance on long-term memory datasets. The RMSE for the Arfima task improved from 1.1521 (RNN) to 0.0861 (ours). Our approach's state forwarding accumulated and transferred states from long-term memory into the current state in an adaptive manner, aided by PSO optimization of the enhanced degree. Ablation study results in Table 5.7 confirmed our method's effectiveness in capturing long-term dependencies.

5.5.4 Comparison With Existing Hardware Reservoirs

Our proposed memristive TDR was compared with other hardware reservoirs based on predictive performance (Table 5.8) and hardware performance (Table 5.10).

Table 5.8: Prediction performance comparison of different models

Datasets	Nar.10	Nar.20	SantaA	SantaD	Hénon	Average (AvRan.*)	Non. audio	Tree	DJI	Arfima	DGR (1-ACC)	Average (AvRan.)	Average (AvRan.)
Type	Short-memory						Long-memory						S&L
Software models													
RNN [85]	0.0403	0.0623	0.0756	0.0356	0.5232	0.1603(8)	0.0254	0.2701	0.2598	1.1521	0.0537	0.3522(7)	0.2485(8)
LSTM [241]	0.0397	0.0497	0.0536	0.0622	0.2298	0.1474(5)	0.0481	0.2788	0.2402	1.1148	0.0858	0.3535(8)	0.2185(6)
mRNN [354]	0.0212	0.0511	0.0240	0.0419	0.2193	0.0715(4)	0.0510	0.2719	0.2396	1.0723	0.0467	0.3303(5)	0.2009(5)
mLSTM [354]	0.0494	0.0502	0.0312	0.0498	0.3597	0.1081(6)	0.0178	0.2703	0.2467	1.1112	0.1106	0.3513(6)	0.2331(7)
ESN [128]	0.0682	0.0689	0.0505	0.0588	0.0311	0.0555(2)	0.0565	0.1209	0.2059	0.1034	0.1567	0.1231(4)	0.0893(3)
Deep-esn [182]	0.0891	0.0885	0.0973	0.0721	0.2512	0.1196(7)	0.0565	0.1286	0.1165	0.0932	0.0762	0.0901(3)	0.1048(4)
TDR [8]	0.0566	0.0671	0.0410	0.0753	0.0596	0.0599(3)	0.0252	0.1286	0.1214	0.1036	0.0616	0.0881(2)	0.0740(2)
Hardware reservoir													
Zhong's [357]	0.0980	0.0907	0.0741	0.0463	0.0192	0.0656(4)	0.0117	0.2762	0.2391	1.150	0.1200	0.3594(4)	0.2125(4)
Bai's [16]	0.0683	-	-	-	-	-	-	-	-	-	-	-	-
Proposed work in Section 4.2.1*	0.0301	0.0476	0.0593	0.0672	0.0488	0.0506(3)	0.0452	**0.0739**	0.0754	0.0885	0.1225	0.08185(3)	0.0645(3)
Proposed work in Section 4.2.2*	0.0862	0.0885	0.0892	0.0928	0.0743	0.0862(5)	0.1370	0.1831	0.1955	0.2096	0.1311	0.1712(5)	0.1287(5)
Proposed work in Section 5.3.4*	0.0372	0.0463	0.0170	0.0237	**0.0046**	0.0257(2)	0.0097	0.1212	0.1096	**0.0860**	0.0744	0.0801(2)	0.0529(2)
Ours	**0.0103**	**0.0420**	**0.00013**	**0.0197**	0.0058	**0.0158(1)**	**0.0093**	0.1106	**0.0107**	0.0861	**0.0366**	**0.0505(1)**	**0.0331(1)**

* AvRan. is the average ranking of the model across multiple datasets, representing its overall performance.

* In order to ensure fair comparisons between our proposed works in Sections 4.2.1, 4.2.2, 5.2.1 (works [260, 261, 264]) and our newly proposed TDR with dual memory enhancement, the number of their reservoir states are set as the same.

Table 5.9: The results of Mann–Whitney U tests with SOTA models and our newly proposed approach

Method	RNN	LSTM	mRNN	mLSTM	Work proposed in Section 4.2.1 [260]
P-value	0.0071	0.0029	0.0070	0.0051	0.0005
Method	ESN	Deep-ESN	TDR	Work proposed in Section 4.2.2 [261]	Work proposed in Section 5.3.4
P-value	0.0046	0.0014	0.0086	0.0106	0.1365

Table 5.10: Comparison between our proposed method and other existing hardware reservoirs

Work	Input layer			Reservoir			Output layer			Total Power	How the models change to adapt to different tasks?
	Type	Anal. Imp.?	Power	Type	Anal. Imp.?	Power	Type	Anal. Imp.?	Power		
Moon's [205]	Random mask	No	CPU power	Basic TDR	Yes (Mem. cir.)	300 μW	Weight matrix	No	CPU power	CPU power $+300\mu W$	Hyperparameter manual-tuning
Zhong's [357]	Random mask	No	CPU power	Basic TDR	Yes (Mem. cir.)	19.2 μW	Weight matrix	No	CPU power	CPU power $+19.2\ \mu W$	Hyperparameter manual-tuning
Wen's [326]	Random mask	No	CPU power	Basic ESN	No	CPU power	Weight matrix	Yes (Mem. cir.)	-	-	Hyperparameter manual-tuning
Zhong's [358]	Random mask	No	CPU power	Basic TDR	Yes (Mem. cir.)	19.2 μW	Weight matrix	Yes (Mem. cir.)	0.0002 μW	CPU power $+19.2\mu W\ \mu W$	Hyperparameter manual-tuning
4T1R work in Section 4.2.1	Random mask	No	CPU power	Reservoir based on 4T1R arch.	Yes (Mem. cir.)	$0.058\mu W$	Weight matrix	No	CPU power	CPU power $+0.058\mu W$	Memory adaptation on-chip
1R work in Section 4.2.2	Random mask	No	CPU power	Reservoir based on 1R arch.	Yes (Mem. cir.)	$0.027\mu W$	Weight matrix	No	CPU power	CPU power $+0.027\mu W$	Memory adaptation on-chip
TDR work in Section 5.3.4	Random mask	No	CPU power	TDR with mem. enh.	Yes (Mem. cir.)	$22.6\mu W$	Weight matrix	No	CPU power	CPU power $+22.6\mu W$	Memory adaptation on-chip
Ours	**Dynamic mask**	**Yes (Mem. cir.)**	**20**μW	**TDR with two mem. enh.**	**Yes (Mem. cir.)**	**22.6**μW	**Weight matrix**	**Yes (Mem. cir.)**	**0.0067** μW	**45.6**μW	**Memory adaptation on-chip**

The works proposed in Sections 4.2.1, 4.2.2, and 5.3.4 have been concluded into corresponding papers [260, 261, 264].

Our hardware implementation outperformed other hardware reservoirs on all datasets in terms of RMSE, as presented in Table 5.8. Moreover, our hardware implementation achieved competitive RMSE with the software implementations discussed in Section 5.5.3. Table 5.10 provides a summary of our memristor-based memory-enhanced TDR design characteristics, along with those of other memristive RC circuits [205, 357, 326, 358, 264]. Regarding the input layer, other SOTA memristive reservoirs have applied random masking as their input layer, and they have not been implemented by analog circuits. Compared with other work, our proposed work has proposed and applied dynamic masking as the input layer. Besides that, we have also used memristive analog circuits to implement this dynamic masking.

As for the reservoir layer, works [205, 357, 358] have applied the memristive circuit to implement the basic TDR function. And one work [326] has not proposed the circuit implementation to the reservoir part of ESN. Compared with other SOTA works, our proposed reservoir not only applied TDR with memory enhancement but also developed its memristive analog implementation, although work [264] has applied a TDR with memory

enhancement by the memristive analog circuits. In terms of the output layer, some of the SOTA works [205, 357, 264] have not proposed their circuit implementation of the weight matrix, and they rely on the PC to post-processing the reservoir state and train the output weight matrix. We applied the same method of calculating the power consumption as [358]. In terms of the power of the reservoir, the power of our work is $22.6\mu W$. Ours is higher than Zhong's works [357, 358], but lower than the CPU implementations [205, 326].

Compared with other SOTA memristive reservoirs, our proposed work also proposed analog circuit implementation for the output layer. Regarding the way how models change to adapt to different tasks, other works rely on the hyperparameter manual tuning, while our work can adapt the memory on-chip for the changes. In terms of the power consumption of the output layer, the power consumption of our work is $0.0067\mu W$. Ours is higher than Zhong's works [358], but lower than the CUP implementations [205, 205, 264]. We also compare our approach with our previous works proposed in Sections 4.2.1 , 4.2.2 and Section 5.3.4 (works [260, 261, 264]). In terms of the reservoir implementation, two previous approaches proposed in Sections 4.2.1 and 4.2.2 have much lower energy consumption than our newly proposed one, because they use regular 4T1R or 1R memristor-based crossbars to construct the reservoir. However, due to this concise and regular architecture of reservoir, their predictive performance is limited as discussed in Section 5.5.3. Compared with the work proposed in Section 5.3.4, our newly proposed approach is more energy efficient due to its fully analog implementation architecture without the CPU power.

Our proposed work involves a fully analog circuit implementation for the input layer, reservoir, and output layer, which is beneficial to the energy efficiency of TDR circuit implementation. Furthermore, our circuit is able to adapt to changing environments on-chip through the use of memory, eliminating the need for manual tuning of hyperparameters. Even though the power consumption is higher than Zhong's works [357], our approach has memory enhancements targeted at improving predictive performance, as presented in Ta-

ble 5.8.

As shown in Figure 5.14 (a), the RMSE performance of digital version of our proposed model is better than that of the analog version over the 11 datasets. However, the power of the digital version is over 30000 times larger than that of the analog version. We choose Intel(R) Xeon(R) W-2235 CPU for estimating the power consumption of the digital version of our proposed model.Figure 5.14 (b) shows RMSE and power comparisons of TDR without memory enhancement VS the TDR with the memory enhancement. As shown in Figure 5.14 (b), the RMSE of the TDR with the memory enhancement outperforms that of the TDR without the memory enhancement over the 11 dataset. While the power of the TDR with the enhanced memory is $45.6\mu W$, which is only twice times of the one without the enhanced memory.

In summary, the implementation of the memory enhancement function will incur an increase in power in the analog circuits. However, compared with the large power consumption of the digital implementation, the increased power consumption caused by the memory enhancement is acceptable and the performance improvement by the memory enhancement is also significant.

5.6 Summary and Discussion

The objective of this chapter was to enhance the memory properties of Time Delay Reservoir (TDR), a promising application field for evolvable hardware, in order to improve its accuracy and energy efficiency. To achieve this, RQ3 was addressed: *"RQ3: How to improve the accuracy and energy efficiency of EHW for RC?"*.

To investigate this research question, we proposed two novel memristive TDR models

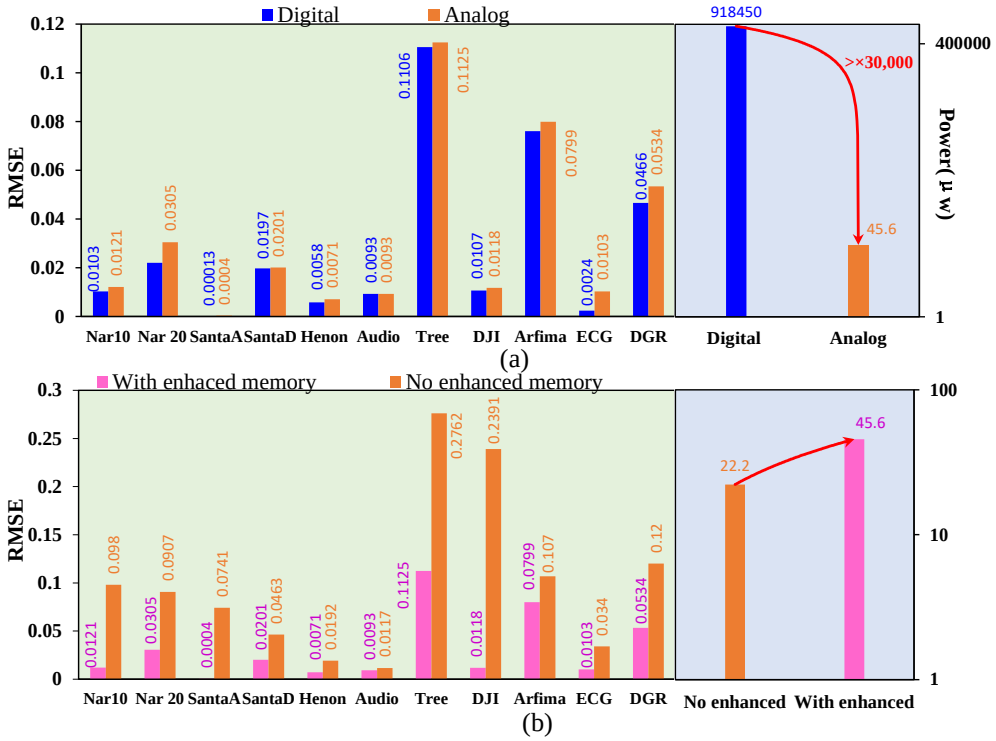


Figure 5.14: (a) The impacts of different types of circuit implementation; (b) The impacts of memory enhancement.

with memory enhancements: the adaptive memory-enhanced memristive TDR through reservoir states and the fully analog memristive TDR with dual memory enhancement. Specifically, the novel contributions of this chapter include:

- A novel high-order time delay unit for TDR, which accumulates and transfers long-term history states, thus enhancing the memory of TDR. The order of the time delay units can be optimized to adjust to different tasks, implementing the adaptive memory enhancement.

- A hardware implementation of the proposed TDR, which is based on two different types of memristor. A single memristor with fade effect is used to construct nonlinear node to exhibit its nonlinear dynamic behavior with short-term memory for TDR. The memristors with nonvolatile memory are used to construct the programmable memristor-based

delay element to enhance the memory of TDR. Moreover, combined with the nonlinear node and the delay element, the memristive implementation framewrok is further proposed.

- We conducted experiments to validate our proposed TDR with adaptive memory enhancement and its memristive implementation. In terms of the prediction performance, our proposed adaptive memory-enhanced TDR obtained better predictive performance than 11 SOTA models on both short-term (5 tasks) and long-term memory datasets (4 tasks) by making use of the adaptive higher-order delay unit, where the memory of TDR can be enhanced and the enhanced degree for each layer can be optimized adaptively according to different given tasks using PSO. We also compared our proposed memristive reservoir implementation with 9 other different hardware reservoirs, where our proposed one outperformed these 9 works, since it only applied one memristor to construct the nonlinear node, further incurring greater energy efficiency.

- A novel TDR with dual memory enhancements that contain two parts to handle short-term and long-term dependencies, respectively. The first part employs a novel dynamic input masking strategy to exploit the historical information of the input time series for handling short-term dependencies. The second part uses the higher-order time delay unit proposed in Section 5.3.4 [264] to handle long-term dependencies, where the reservoir state forwarding technique is used to strengthen the interaction and accumulation of long-term states

- Moreover, we also presented a fully analog memristive circuit design of our proposed TDR with dual memory enhancements, which leverages the hardware-friendly features of TDRs and boosts the energy efficiency of a fully analog circuit design. In the proposed circuit design, we used two types of memristors to construct the input layer, reservoir layer, and output layer, where a dynamic memristor is used as a nonlinear node and nonvolatile memristor is used to construct memristor-based window element

and memristor-based delay element.

- We also conducted experiments to verify our proposed TDR with dual memory enhancements and its fully analog memristive implementation. As shown experimentally, dynamic masking helps the reservoir focus on the most important information, improving its ability to store and recall short-term memory. State forwarding helps the reservoir maintain a sense of context, improving its ability to store and recall long-term memory. We also compared our proposed approach with 12 SOTA models on 5 short-term memory datasets and 5 long-term memory datasets. In terms of predictive performance, the experimental results show that our proposed implementation outperformed those existing 12 works across on both of the short-term and long-term datasets. Moreover, our proposed fully analog memristive implementation outperformed other 7 reservoir hardware implementations in terms of power consumption, as our proposed one has the fully analog memristive implementation including the input, reservoir and output layers.

Chapter Six

Conclusions and Future Work

The development of memristor-based EHW is a promising but challenging research direction. The primary objective of this thesis has been to advance the development of EHW using memristors, through the improvement of circuit representations, the promotions of evolutionary processes, and the expansion of memristive circuit applications. This chapter summarises the contributions of this thesis (Section 6.1) and discusses the potential direction of the future work (Section 6.2).

6.1 Conclusion

EHW based on memristors is a promising but challenging research area, requiring a comprehensive investigation from both hardware and algorithm perspectives. In this thesis, three research questions were raised to address key challenges in this field. First, we addressed the problems of how to represent and evolve analog circuits for EHW. We proposed a novel tree-based circuit representation that enables a more efficient and general circuit encoding method than the existing approaches. Based on this representation, we designed a novel circuit evolution algorithm that can better support the analog circuit evolution by using a

Shapely value based evolutionary operations and a two-stage evolutionary framework.

Second, besides providing a solution to overcome the above mentioned problems of existing work, when introducing memristors to EHW, there are also some new problems, such as how to design indirect circuit representations and corresponding evolutionary algorithms for different memristive reconfigurable architectures. Therefore, we investigated how to design indirect circuit representations and evolutionary algorithms for different types of memristive reconfigurable architectures, such as 4T1R architecture and 1R architecture. We demonstrated that the proposed memristive EHW can implement memristive reservoir computing, which is a powerful technique for processing time series data, and successfully adapt to various tasks.

Third, we demonstrated the practicality and applicability of our proposed novel memristor-based EHW approach by successfully applying it to real-world applications, specifically through reservoir computing techniques. In particular, we proposed novel memristive EHWs that not only enhance the accuracy of time series processing but also improve the energy efficiency of hardware computing systems by using adaptive delay feedback, and optimizing memristive circuits.

Specifically, the contributions of this thesis include the following aspects:

- **Circuit Evolutionary Approach with Improved Generality and Efficiency:** The design of the evolutionary algorithm plays a significant role in developing EHW. Existing algorithms have limitations, such as the limited design generality of the circuit representation and inefficient evolutionary process [38, 151]. Chapter 3 proposed a novel tree-based circuit representation and more efficient evolutionary algorithm to tackle the problems of existing algorithms for EHW based on memristors, answering the RQ1 of the thesis:

RQ1: How to improve general applicability and efficiency of the circuit evolution?

Chapter 3 answered RQ1 from both of the circuit representation and evolutionary process perspectives. A new tree-based representation was proposed, which addresses the challenges of existing circuit representations, such as limited design capabilities (generality) [38] and inefficient transformation to circuit netlists [151]. The proposed circuit representation has improved generality, being compatible with both two-terminal and three-terminal circuits. It can also be transformed more directly and efficiently into circuit netlists. Additionally, this new representation also proposes a suitable crossover operator that better supports automated circuit evolution. Moreover, Chapter 3 also proposed a novel evolutionary algorithm, which employs a novel strategy based on Shapley values and a two-stage evolutionary framework to evaluate the contribution of each function node in a circuit and guide the search towards more promising regions of the search space. These techniques enable more efficient and general design of EHWs using memristors.

- **Memristive Circuit Evolution Based on Reconfigurable Architectures**:
  Memristor is an emerging devicethat is promising to be applied to EHWs. However, applying it to the EHW design involves some challenges related to the design of the memristor-based EHW, such as designing new indirect circuit representations and corresponding evolutionary operators for different memristor-based reconfigurable architectures. Therefore, the circuit representations of the memristive reconfigurable architectures should be designed first, which provides the encoding method of the memristor-based EHW. We have investigated the indirect circuit representation and the evolutionary algorithms for different memristive reconfigurable architectures. Chapter 4 proposed two indirect circuit representations and their corresponding evolutionary operators for two different reconfigurable memristor-based architectures, 4T1R and 1R architectures, respectively, answering the RQ2 of the thesis:

RQ2: How to evolve memristive circuits based on reconfigurable architectures?

In Chapter 4, we proposed an indirect circuit representation and its corresponding evolutionary operators for 4T1R memristive reconfigurable architecture, which enables the implementation of memristive reservoir computing on this EHW paradigm. This representation evolves the configuration signals of the 4T1R memristive unit, rather than the memristor state, and records the current states flowing through the memristor as the reservoir states. This can avoid the variances caused by the variation between the actual and ideal memristors. Moreover, this representation takes into account the circuit feasibility and scalability, by using sparse initialization and evolutionary operators designed for memristive reservoir circuits. The 4T1R memristive active array has the advantage of allowing more precise and flexible control of the memristor state and more diverse dynamic behaviors of the array, but the drawback of increasing the circuit overhead, as it requires four transistors and one memristor per cell.

We also developed a novel indirect circuit representation and its tailored evolutionary operators for 1R memristive reconfigurable architecture, which facilitates the realization of memristive reservoir computing on this EHW paradigm. In this representation, we exploit the sneak currents of the 1R memristor-based crossbar for reservoir computing, instead of avoiding them as they are usually considered as adverse effects that interfere with the write/read operation. For this purpose, we devised an evolutionary algorithm with innovative evolutionary operators that can spatiotemporally adapt the memristive reservoir. The memristor states, input-connection states, and input-injection timing of the memristive crossbar are evolved on the 1R memrsitor-based reconfigurable architecture, leading to the formation of adaptive mask and modularity in the reservoir. The 1R memristive passive array has the benefit of reducing the circuit overhead, as it only requires one memristor per cell, but the challenge of making the control of the memristor state more difficult, as it depends on the resistance of the

other cells in the same row and column.

- **Accurate and Energy-efficient EHW Design for Reservoir Computing**:
  Reservoir computing still suffers several restrictions from both algorithm and hardware perspectives. In terms of the algorithm part, standard TDRs have a limited and inflexible memory property to store and retrieve information over time [8, 264]. In terms of the hardware part, existing hardware implementations of reservoir computing still rely on digital or partially digital components, which could incur larger power consumption. In Chapter 5, two memristive TDRs with different memory enhancement techniques were proposed, answering the RQ3 of the thesis:

  RQ3: How to improve the accuracy and energy efficiency of EHW for RC?

  Chapter 5 first introduced a higher-order delay unit, which is capable of accumulating and transferring the long history states in an adaptive manner to further enhance the reservoir memory. Particle Swarm Optimisation was applied to optimize the enhanced degree of memory adaptivity. We further proposed a memristive implementation of our adaptive memory-enhanced TDR, where a dynamic memristor and the memristor-based delay element were applied to construct the reservoir. With this design our proposed adaptive memory-enhanced TDR obtained better predictive performance on both short-term and long-term memory datasets. By virtue of nano-size and low energy efficiency of memristor, the proposed memristive implementation also showed its superiority on the circuit area and power consumption compared with traditional device-based reservoirs.

  Based on the aforementioned work, Chapter 5 further proposed a novel TDR with dual enhanced memory and its fully analog memristive circuit design to boost the power efficiency of the hardware implementation of the proposed TDR. Specifically, two parts were designed: (1) dynamic masking to strengthen the interaction between adjacent time steps with a moving and adaptable window; and (2) reservoir state forwarding to

accumulate and jump between long-term states adaptively. We also designed a fully analog memristive circuit for our TDR with memory enhancement, using two types of memristors for the input, reservoir, and output layers. Experiments showed that our TDR and its memristive implementation are feasible and effective for short- and long-term memory datasets, and consumed less power than conventional or digital reservoirs.

6.2 Future Work

In this thesis, we have made several contributions to the field of EHWs using emerging electronic devices. However, there are still some open challenges and opportunities for further research. In this section, we outline some possible directions for future work.

- **Scalability of Memristor-based EHW**

  One of the main challenges in the EHW research is the scalability problem, which hinders the development of EHW. In this thesis, we have made our primary attempts to tackle this scalability issue of memristor-based EHW, where we proposed a novel circuit representation with more efficient transformation into the circuit netlists, and a novel evolutionary algorithm with more efficient evolutionary process. In the future, this problem can be further addressed from both the algorithmic and hardware perspectives. On the algorithmic side, the current approach relies on accurate circuit simulations to evaluate the performance of circuit individuals, which can be costly and limit the scalability in terms of time efficiency. Therefore, machine learning techniques could be employed to estimate the performance of evolved circuits and guide the evolutionary process towards more optimal solutions. For instance, neural networks could be used to predict the performance of a circuit based on its topology, enabling more

efficient evaluation and selection of candidate solutions. On the hardware side, more parallel hardware implementations could be explored to further enhance the scalability of memristor-based EHW.

- **Explainability of Memristor-based EHW**

  Memristor-based EHW have been demonstrated as a powerful computing architecture for a wide range of AI applications and are considered to be the next generation of artificial intelligence. However, the lack of model explainability in existing memristive EHW makes it difficult to properly interpret decision results, resulting in unreliable, unaccountable, and untrustworthy decisions. To address this issue, we will investigate the explainability of memristor-based EHW to further advance the understanding of these hardware computing systems in the future. We will explore various methods and techniques to extract and present the relevant features and factors that influence the decision-making process of memristive EHW. We will also evaluate the quality and usefulness of the explanations generated by our approaches on different tasks and domains. Our goal is to provide more transparent, interpretable, and trustworthy memristive EHW for AI applications. We have made our first attempt to study the explainability of memristor-based EHW in our published paper [**2] listed in Chapter 1.

- **Hardware Implementability of Memristor-based EHW**

  One of the main directions for future work is to enhance the hardware implementability of memristor-based EHW from the device level and array level by evolutionary approaches. In our current proposed work, we assumed ideal memristor characteristics and ignored non-idealities in our circuit simulation. In the actual physical hardware implementation, memristors are nonlinear devices that exhibit complex and diverse behaviors, such as write-to-write variation and device-to-device variations. These behaviors can pose challenges or opportunities for different applications and tasks, de-

pending on the desired functionality and performance. Therefore, it is important to understand and control the non-ideality of memristor from both the device level and array level. In the future, we will incorporate these non-idealities into our research and explore various evolutionary approaches that can leverage them for different purposes. For example, we will investigate the algorithms that are more tolerant to these non-idealities of memristor and further exploit them to achieve desired non-ideal behaviors. We will also investigate how to evolve the array topology and configuration to cope with the non-ideal effects, such as sneak currents and stray capacitance. We will evaluate the effectiveness and efficiency of our evolutionary approaches on different tasks and domains, such as neuromorphic computing and other more complex applications.

- **On-Chip Evolvability of Memristor-based EHW**

  Another future work is to study the on-chip evolvability of the memristor-based EHW. This will enable us to achieve more autonomous and efficient EHW systems that can adapt to dynamic and uncertain environments by themselves. However, the evolutionary algorithms of our proposed methods EHW based on memristor are still run on the extrinsic PC. Therefore, we will explore various mechanisms and strategies to enhance the intrinsic evolvability of the memristor-based EHW, such as exploiting the memristive dynamics and plasticity, incorporating learning and evolution, developing self-organization and self-repair capabilities, and utilizing local and distributed feedback and communication. We will also investigate how to measure and monitor the intrinsic evolutionary progress and performance of the EHW based on memristor, as well as how to provide feedback and guidance to the self-evolution process.

References

[1] M Taher Abuelma'Atti. "Universal CMOS current-mode analog function synthesizer". In: *IEEE Transactions on Circuits & Systems I: Regular Papers* 49.10 (2011), p.1468–1474.

[2] Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. "PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture". In: *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. 2015, pp. 336–348.

[3] Shawkat Ali, Saleem Khan, Arshad Khan, and Amine Bermak. "Memristor fabrication through printing technologies: A review". In: *IEEE Access* 9 (2021), pp. 95970–95985.

[4] Miquel L Alomar, Vincent Canals, Nicolas Perez-Mora, Víctor Martínez-Moll, and Josep L Rosselló. "FPGA-based stochastic echo state networks for time-series forecasting". In: *Computational Intelligence And Neuroscience* 2016 (2016), pp. 1–15.

[5] Pablo Amil, Cecilia Cabeza, and Arturo C Marti. "Exact discrete-time implementation of the Mackey–Glass delayed model". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 62.7 (2015), pp. 681–685.

[6] S. Ando and H. Iba. "Analog circuit design with a variable length chromosome". In: *Proceedings of the 2000 Congress on Evolutionary Computation. (CEC)*. Vol. 2. 2000, pp. 994–1001.

[7] Takafum Aoki, Naofumi Homma, and Tatsuo Higuchi. "Evolutionary synthesis of arithmetic circuit structures". In: *Artificial Intelligence Review* 20.3-4 (2003), pp. 199–232.

[8] Lennert Appeltant, Miguel Cornelles Soriano, Guy Van der Sande, Jan Danckaert, Serge Massar, Joni Dambre, Benjamin Schrauwen, Claudio R Mirasso, and Ingo Fischer. "Information processing using a single dynamical node as complex system". In: *Nature Communications* 2.1 (2011), pp. 1–6.

[9] J Scott Armstrong and Fred Collopy. "Error measures for generalizing about forecasting methods: Empirical comparisons". In: *International journal of forecasting* 8.1 (1992), pp. 69–80.

[10] Amir F Atiya and Alexander G Parlos. "New results on recurrent network training: unifying the algorithms and accelerating convergence". In: *IEEE Transactions on Neural Networks and Learning Systems* 11.3 (2000), pp. 697–709.

[11] Farooq Azam. "Biologically inspired modular neural networks". PhD thesis. Virginia Polytechnic Institute and State University, 2000.

[12] Mostafa Rahimi Azghadi, Bernabe Linares-Barranco, Derek Abbott, and Philip HW Leong. "A hybrid CMOS-memristor neuromorphic synapse". In: *IEEE Transactions on Biomedical Circuits and Systems* 11.2 (2016), pp. 434–445.

[13] Woorham Bae, Kyung Jean Yoon, Cheol Seong Hwang, and Deog-Kyoon Jeong. "A crossbar resistance switching memory readout scheme with sneak current cancellation based on a two-port current-mode sensing". In: *Nanotechnology* 27.48 (2016), p. 485201.

[14] Woorham Bae, Kyung Jean Yoon, Taeksang Song, and Borivoje Nikolić. "A variation-tolerant, sneak-current-compensated readout scheme for cross-point memory based

on two-port sensing technique". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 65.12 (2018), pp. 1839–1843.

[15] Seungbum Baek. "Reconfigurable multiplier architecture based on memristor-cmos with higher flexibility". In: *arXiv preprint arXiv:1907.09078* (2019).

[16] Kangjun Bai and Yang Yi. "DFR: An energy-efficient analog delay feedback reservoir computing system for brain-inspired computing". In: *ACM Journal on Emerging Technologies in Computing Systems* 14.4 (2018), pp. 1–22.

[17] Abubakar Bala, Idris Ismail, Rosdiazli Ibrahim, and Sadiq M Sait. "Applications of metaheuristics in reservoir computing techniques: a review". In: *IEEE Access* 6 (2018), pp. 58012–58029.

[18] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming*. Springer, 1998.

[19] Zhiguo Bao and Takahiro Watanabe. "A new approach for circuit design optimization using genetic algorithm". In: *2008 International SoC Design Conference (ISOCC)*. Vol. 1. 2008, pp. I–383.

[20] Davide Basilio Bartolini, Fabio Cancare, Matteo Carminati, and Donatella Sciuto. "Hera: Hardware evolution over reconfigurable architectures". In: *2011 1st International Workshop on Computing in Heterogeneous, Autonomous' N'Goal-Oriented Environments (CHANGE)*. 2011, pp. 1–8.

[21] Daniel Batas and Horst Fiedler. "A memristor SPICE implementation and a new approach for magnetic flux-controlled memristor modeling". In: *IEEE Transactions on Nanotechnology* 10.2 (2010), pp. 250–255.

[22] Michael Benedicks and Lennart Carleson. "The dynamics of the Hénon map". In: *Annals of Mathematics* (1991), pp. 73–169.

[23] Forrest H Bennett, John R Koza, David Andre, and Martin A Keane. "Evolution of a 60 decibel op amp using genetic programming". In: *International Conference on Evolvable Systems (ICS)*. 1996, pp. 453–469.

[24] Jan Beran, Yuanhua Feng, Sucharita Ghosh, and Rafal Kulik. *Long-Memory Processes.* Springer, 2016.

[25] Radu Berdan, Takao Marukame, Kensuke Ota, Marina Yamaguchi, Masumi Saitoh, Shosuke Fujii, Jun Deguchi, and Yoshifumi Nishi. "Low-power linear computation using nonlinear ferroelectric tunnel junction memristors". In: *Nature Electronics* 3.5 (2020), pp. 259–266.

[26] Filippo Maria Bianchi, Simone Scardapane, Sigurd Løkse, and Robert Jenssen. "Reservoir computing approaches for representation and classification of multivariate time series". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[27] Adria Bofill-i-Petit and Alan F Murray. "Synchrony detection and amplification by silicon neurons with STDP synapses". In: *IEEE Transactions on Neural Networks and Learning Systems* 15.5 (2004), pp. 1296–1304.

[28] George EP Box and David A Pierce. "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models". In: *Journal of the American Statistical Association* 65.332 (1970), pp. 1509–1526.

[29] Stephen P Boyd, Thomas H Lee, et al. "Optimal design of a CMOS op-amp via geometric programming". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20.1 (2001), pp. 1–21.

[30] Rodica Branzei, Dinko Dimitrov, and Stef Tijs. *Models in cooperative game theory.* Vol. 556. Springer Science & Business Media, 2008.

[31] A Paul Brokaw. "A simple three-terminal IC bandgap reference". In: *IEEE Journal of Solid-State Circuits* 9.6 (1974), pp. 388–393.

[32] Daniel Brunner, Miguel C Soriano, Claudio R Mirasso, and Ingo Fischer. "Parallel photonic information processing at gigabyte per second data rates using transient states". In: *Nature Communications* 4.1 (2013), pp. 1–7.

[33] Jens Bürger, Alireza Goudarzi, Darko Stefanovic, and Christof Teuscher. "Hierarchical composition of memristive networks for real-time computing". In: *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2015, pp. 33–38.

[34] Jens Bürger and Christof Teuscher. "Variation-tolerant computing with memristive reservoirs". In: *Proceedings of the 9th IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2013, pp. 1–6.

[35] Juan Pablo Carbajal, Joni Dambre, Michiel Hermans, and Benjamin Schrauwen. "Memristor models for machine learning". In: *Neural Computation* 27.3 (2015), pp. 725–747.

[36] Federico Castejón and E Carmona. "Automatic design of electronic amplifiers using grammatical evolution". In: *Actas de Multiconferencia CAEPIA-13* (2013), pp. 703–712.

[37] Federico Castejón and Enrique J Carmona. "Automatic design of analog electronic circuits using grammatical evolution". In: *Applied Soft Computing* 62 (2018), pp. 1003–1018.

[38] Shoou-Jinn Chang, Hao-Sheng Hou, and Yan-Kuin Su. "Automated passive filter synthesis using a novel tree representation and genetic programming". In: *IEEE Transactions on Evolutionary Computation* 10.1 (2006), pp. 93–100.

[39] André Chanthbouala, Vincent Garcia, Ryan O Cherifi, Karim Bouzehouane, Stéphane Fusil, Xavier Moya, Stéphane Xavier, Hiroyuki Yamada, Cyrile Deranlot, Neil D

Mathur, et al. “A ferroelectric memristor”. In: *Nature Materials* 11.10 (2012), pp. 860–864.

[40] Kyriakos C Chatzidimitriou and Pericles A Mitkas. “Adaptive reservoir computing through evolution and learning”. In: *Neurocomputing* 103 (2013), pp. 198–209.

[41] Dingjun Chen, Takafumi Aoki, Naofumi Homma, Toshiki Terasaki, and Tatsuo Higuchi. “Graph-based evolutionary design of arithmetic circuits”. In: *IEEE Transactions on Evolutionary Computation* 6.1 (2002), pp. 86–100.

[42] Huanhuan Chen, Peter Tiňo, Ali Rodan, and Xin Yao. “Learning in the model space for cognitive fault diagnosis”. In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2013), pp. 124–136.

[43] Jia Chen, Jiancong Li, Yi Li, and Xiangshui Miao. “Multiply accumulate operations in memristor crossbar arrays for analog computing”. In: *Journal of Semiconductors* 42.1 (2021), p. 013104.

[44] Ling Chen, Chuandong Li, Tingwen Huang, Yiran Chen, Shiping Wen, and Jiangtao Qi. “A synapse memristor model with forgetting effect”. In: *Physics Letters A* 377.45-48 (2013), pp. 3260–3265.

[45] Ling Chen, Chuandong Li, Tingwen Huang, Xiaofang Hu, and Yiran Chen. “The bipolar and unipolar reversible behavior on the forgetting memristor model”. In: *Neurocomputing* 171 (2016), pp. 1637–1643.

[46] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).

[47] Naima Chouikhi, Boudour Ammar, Nizar Rokbani, and Adel M Alimi. "PSO-based analysis of Echo State Network parameters for time series forecasting". In: *Applied Soft Computing* 55 (2017), pp. 211–225.

[48] Leon Chua. "Memristor-the missing circuit element". In: *IEEE Transactions on Circuit Theory* 18.5 (1971), pp. 507–519.

[49] Stefano Cipriani and Anthony A Takeshian. *Compact cubic function generator*. US Patent 6,160,427. 2000.

[50] Jose M Cruz-Albrecht, Michael W Yung, and Narayan Srinivasa. "Energy-efficient neuron, synapse and STDP integrated circuits". In: *IEEE Transactions on Biomedical Circuits and Systems* 6.3 (2012), pp. 246–256.

[51] Matthew Dale, Richard FL Evans, Sarah Jenkins, Simon O'Keefe, Angelika Sebald, Susan Stepney, Fernando Torre, and Martin Trefzer. "Reservoir computing with thin-film ferromagnetic devices". In: *arXiv preprint arXiv:2101.12700* (2021).

[52] Matthew Dale, Julian F Miller, Susan Stepney, and Martin A Trefzer. "Evolving carbon nanotube reservoir computers". In: *Unconventional Computation and Natural Computation: 15th International Conference, UCNC 2016, Manchester, UK, July 11-15, 2016, Proceedings 15*. Springer. 2016, pp. 49–61.

[53] Matthew Dale, Susan Stepney, Julian F Miller, and Martin Trefzer. "Reservoir computing in materio: An evaluation of configuration through evolution". In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2016, pp. 1–8.

[54] Angan Das and Ranga Vemuri. "A graph grammar based approach to automated multi-objective analog circuit design". In: *2009 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2009, pp. 700–705.

[55] Angan Das and Ranga Vemuri. "An automated passive analog circuit synthesis framework using genetic algorithms". In: *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 2007, pp. 145–152.

[56] Tathagato Rai Dastidar, PP Chakrabarti, and Partha Ray. "A synthesis system for analog circuits based on evolutionary search and topological reuse". In: *IEEE Transactions on Evolutionary Computation* 9.2 (2005), pp. 211–224.

[57] Hugo De Garis and Michael Korkin. "The CAM-Brain Machine (CBM): an FPGA-based hardware tool that evolves a 1000 neuron-net circuit module in seconds and updates a 75 million neuron artificial brain for real-time robot control". In: *Neurocomputing* 42.1-4 (2002), pp. 35–68.

[58] K Dejhan and C Netbut. "New simple square-rooting circuits based on translinear current conveyors". In: *International journal of electronics* 94.7 (2007), pp. 707–723.

[59] Yamille Del Valle, Ganesh Kumar Venayagamoorthy, Salman Mohagheghi, Jean-Carlos Hernandez, and Ronald G Harley. "Particle swarm optimization: basic concepts, variants and applications in power systems". In: *IEEE Transactions on Evolutionary Computation* 12.2 (2008), pp. 171–195.

[60] Ronald F DeMara, Kening Zhang, and Carthik A Sharma. "Autonomic fault-handling and refurbishment using throughput-driven assessment". In: *Applied Soft Computing* 11.2 (2011), pp. 1588–1599.

[61] Florian Denis-Le Coarer, Marc Sciamanna, Andrew Katumba, Matthias Freiberger, Joni Dambre, Peter Bienstman, and Damien Rontani. "All-optical reservoir computing on a photonic chip using silicon-based ring resonators". In: *IEEE Journal of Selected Topics in Quantum Electronics* 24.6 (2018), pp. 1–8.

[62] Kudithipudi Dhireesha, Saleh Qutaiba, Merkel Cory, Thesing James, and Wysocki Bryant. "Design and Analysis of a Neuromemristive Reservoir Computing Architecture for Biosignal Processing". In: *Frontiers in Neuroscience* 9 (2016), p. 502.

[63] Robert DiPietro, Christian Rupprecht, Nassir Navab, and Gregory D Hager. "Analyzing and exploiting NARX recurrent neural networks for long-term dependencies". In: *arXiv preprint arXiv:1702.07805* (2017).

[64] Mian Dong and Lin Zhong. "Nanowire crossbar logic and standard cell-based integration". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 17.8 (2009), pp. 997–1007.

[65] Ronald G Dreslinski, Michael Wieckowski, David Blaauw, Dennis Sylvester, and Trevor Mudge. "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits". In: *Proceedings of the IEEE* 98.2 (2010), pp. 253–266.

[66] Chao Du, Fuxi Cai, Mohammed A Zidan, Wen Ma, Seung Hwan Lee, and Wei D Lu. "Reservoir computing using dynamic memristors for temporal information processing". In: *Nature Communications* 8.1 (2017), pp. 1–10.

[67] Wen Du, Caihong Li, Yixuan Huang, Jihua Zou, Lingzhi Luo, Caihong Teng, Hao-Chung Kuo, Jiang Wu, and Zhiming Wang. "An optoelectronic reservoir computing for temporal information processing". In: *IEEE Electron Device Letters* 43.3 (2022), pp. 406–409.

[68] François Duport, Akram Akrout, Anteo Smerieri, Marc Haelterman, and Serge Massar. "Analog input layer for optical reservoir computers". In: *arXiv preprint arXiv:1406.3238* (2014).

[69] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*. Vol. 53. Springer, 2003.

[70] Matthew E Falagas, Eleni I Pitsouni, George A Malietzis, and Georgios Pappas. "Comparison of PubMed, Scopus, web of science, and Google scholar: strengths and weaknesses". In: *The FASEB journal* 22.2 (2008), pp. 338–342.

[71] Wei Fei, Hao Yu, Wei Zhang, and Kiat Seng Yeo. "Design exploration of hybrid cmos and memristor circuit by new modified nodal analysis". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 20.6 (2011), pp. 1012–1025.

[72] Aida A Ferreira and Teresa B Ludermir. "Evolutionary strategy for simultaneous optimization of parameters, topology and reservoir weights in Echo State Networks". In: *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. 2010, pp. 1–7.

[73] Aida A Ferreira, Teresa B Ludermir, and Ronaldo RB De Aquino. "An approach to reservoir computing design and training". In: *Expert Systems with Applications* 40.10 (2013), pp. 4172–4182.

[74] Robyn Ffrancon and Marc Schoenauer. "Memetic semantic genetic programming". In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO)*. 2015, pp. 1023–1030.

[75] Thomas Fischer and Christopher Krauss. "Deep learning with long short-term memory networks for financial market predictions". In: *European Journal of Operational Research* 270.2 (2018), pp. 654–669.

[76] Luca Fossati and Jorgen Ilstad. "The future of embedded systems at ESA: Towards adaptability and reconfigurability". In: *2011 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*. 2011, pp. 113–120.

[77] Keisuke Fujii and Kohei Nakajima. "Harnessing disordered-ensemble quantum dynamics for machine learning". In: *Physical Review Applied* 8.2 (2017), p. 024030.

[78] Zhaohui Gan, Zhenkun Yang, Tao Shang, Tianyou Yu, and Min Jiang. "Automated synthesis of passive analog filters using graph representation". In: *Expert Systems with Applications* 37.3 (2010), pp. 1887–1898.

[79] Alan E Gelfand, Dipak K Dey, and Hong Chang. *Model determination using predictive distributions with implementation via sampling-based methods*. Tech. rep. Department of Statistics, Stanford University, California, 1992.

[80] Suma George, Sihwan Kim, Sahil Shah, Jennifer Hasler, Michelle Collins, Farhan Adil, Richard Wunderlich, Stephen Nease, and Shubha Ramakrishnan. "A programmable and configurable mixed-mode FPAA SoC". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.6 (2016), pp. 2253–2261.

[81] Rahul Gharpinde, Phrangboklang Lynton Thangkhiew, Kamalika Datta, and Indranil Sengupta. "A scalable in-memory logic synthesis approach using memristor crossbar". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.2 (2017), pp. 355–366.

[82] Amirata Ghorbani and James Zou. "Data shapley: Equitable valuation of data for machine learning". In: *International Conference on Machine Learning (ICML)*. 2019, pp. 2242–2251.

[83] Amirata Ghorbani and James Y Zou. "Neuron shapley: Discovering the responsible neurons". In: *Advances in Neural Information Processing Systems (NIPS)* 33 (2020), pp. 5922–5932.

[84] Barrie Gilbert. "A precise four-quadrant multiplier with subnanosecond response". In: *IEEE Journal of Solid-State Circuits* 3.4 (1968), pp. 365–373.

[85] C Lee Giles, Gary M Kuhn, and Ronald J Williams. "Dynamic recurrent neural networks: Theory and applications". In: *IEEE Transactions on Neural Networks and Learning Systems* 5.2 (1994), pp. 153–156.

[86] Kyrre Glette, Jim Torresen, and Moritoshi Yasunaga. "An online EHW pattern recognition system applied to sonar spectrum classification". In: *Evolvable Systems: From Biology to Hardware: 7th International Conference, (ICES)*. Springer. 2007, pp. 1–12.

[87] T. G. W. Gordon and P. J. Bentley. "Development brings scalability to hardware evolution". In: *Nasa/DoD Conference on Evolvable Hardware (EH)*. 2005.

[88] Alireza Goudarzi, Matthew R Lakin, and Darko Stefanovic. "Reservoir computing approach to robust computation using unreliable nanoscale networks". In: *International Conference on Unconventional Computation and Natural Computation (UCNC)*. 2014, pp. 164–176.

[89] CWJ Granger and R Joyeux. "An introduction to long-range time series models and fractional integration". In: *Journal of Time Series Analysis* 1 (1980), pp. 15–30.

[90] Alexander Greaves-Tunnell and Zaid Harchaoui. "A statistical investigation of long memory in language and music". In: *Proceedings of the 36th International Conference on Machine Learning (ICML)*. 2019, pp. 2394–2403.

[91] Lyudmila Grigoryeva, Julie Henriques, Laurent Larger, and Juan-Pablo Ortega. "Optimal nonlinear information processing capacity in delay-based reservoir computers". In: *Scientific Reports* 5.1 (2015), pp. 1–11.

[92] Lyudmila Grigoryeva, Julie Henriques, Laurent Larger, and Juan-Pablo Ortega. "Stochastic nonlinear time series forecasting using time-delay reservoir computers: Performance and universality". In: *Neural Network* 55 (2014), pp. 59–71.

[93] James B Grimbleby. "Automatic analogue network synthesis using genetic algorithms". In: *The 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*. IET. 1995, pp. 53–58.

[94] Ximeng Guan, Shimeng Yu, and H-S Philip Wong. "On the switching parameter variation of metal-oxide RRAM—Part I: Physical modeling and simulation methodology". In: *IEEE Transactions on Electron Devices* 59.4 (2012), pp. 1172–1182.

[95] Ivick Guerra-Gómez, Trent Mcconaghy, and Esteban Tlelo-Cuautle. "Operating-point driven formulation for analog computer-aided design". In: *Analog Integrated Circuits and Signal Processing* 74.2 (2013), pp. 345–353.

[96] Isha Gupta, Alexantrou Serb, Ali Khiat, Ralf Zeitler, Stefano Vassanelli, and Themistoklis Prodromakis. "Real-time encoding and compression of neuronal spikes by metal-oxide memristors". In: *Nature Communications* 7.1 (2016), pp. 1–9.

[97] Simon Harding and Julian F Miller. "Evolution in materio: A tone discriminator in liquid crystal". In: *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*. Vol. 2. IEEE. 2004, pp. 1800–1807.

[98] Ramesh Harjani, Rob A Rutenbar, and L Richard Carley. "OASYS: A framework for analog circuit synthesis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 8.12 (1989), pp. 1247–1266.

[99] M. Hartmann, P. K. Lehre, and P. C. Haddow. "Evolved digital circuits and genome complexity". In: *2005 NASA/DoD Conference on Evolvable Hardware (EH)*. 2005, pp. 79–86.

[100] Amad Ul Hassen. "Automated synthesis of compact multiplier circuits for in-memory computing using ROBDDs". In: *2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2017, pp. 141–146.

[101] Helmut Hauser, Auke J Ijspeert, Rudolf M Füchslin, Rolf Pfeifer, and Wolfgang Maass. "Towards a theoretical foundation for morphological computation with compliant bodies". In: *Biological Cybernetics* 105.5 (2011), pp. 355–370.

[102] Jingsong He, Mingguo Liu, and Yunbi Chen. "A novel real-coded scheme for evolutionary analog circuit synthesis". In: *2009 International Workshop on Intelligent Systems and Applications (ISA)*. 2009, pp. 1–4.

[103] Shan He. "Training Artificial Neural Networks Using Lévy Group Search Optimizer." In: *Journal of Multiple-Valued Logic & Soft Computing* 16.6 (2010).

[104] Shan He, Q Henry Wu, and Jon R Saunders. "Group search optimizer: an optimization algorithm inspired by animal searching behavior". In: *IEEE Transactions on Evolutionary Computation* 13.5 (2009), pp. 973–990.

[105] T. Higuchi. "Evolvable Hardware and its Applications to Pattern Recognition and Fault-tolerant Systems". In: *The 1st International Workshop towards Evolvable Hardware*. 1995.

[106] Tetsuya Higuchi, Masaya Iwata, Isamu Kajitani, Hitoshi Yamada, Bernard Manderick, Yuji Hirao, Masahiro Murakawa, Shuji Yoshizawa, and Tatsumi Furuya. "Evolvable hardware with genetic learning". In: *1996 IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 4. 1996, pp. 29–32.

[107] Tetsuya Higuchi, Yong Liu, and Xin Yao. *Evolvable Hardware (Genetic and Evolutionary Computation)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[108] Tetsuya Higuchi, Masahiro Murakawa, Masaya Iwata, Isamu Kajitani, Weixin Liu, and Mehrdad Salami. "Evolvable hardware at function level". In: *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (CEC)*. 1997, pp. 187–192.

[109] Tetsuya Higuchi, Tatsuya Niwa, Toshio Tanaka, Hitoshi Iba, Hugo de Garis, and Tatsumi Furuya. "Evolving hardware with genetic learning: A first step towards building a Darwin machine". In: *Proceedings of the 2nd International Conference on Simulated Adaptive Behaviour (SAB)*. MIT Press Cambridge. 1993, pp. 417–424.

[110] Sepp Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* 91.1 (1991).

[111] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[112] David Hodan, Vojtech Mrazek, and Zdenek Vasicek. "Semantically-oriented mutation operator in cartesian genetic programming for evolutionary circuit design". In: *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*. 2020, pp. 940–948.

[113] Arthur E Hoerl and Robert W Kennard. "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1 (1970), pp. 55–67.

[114] Gordon Hollingworth, Steve Smith, and Andy Tyrrell. "Safe intrinsic evolution of virtex devices". In: *Proceedings of 2nd NASA/DoD Workshop on Evolvable Hardware (NEHW)*. 2000, pp. 195–202.

[115] Georg Holzmann. "Reservoir computing: a powerful black-box framework for nonlinear audio processing". In: *Proceedings of the 12th International Conference on Digital Audio Effects (DAFx)*. 2009.

[116] Qinghui Hong, Hegan Chen, Jingru Sun, and Chunhua Wang. "Memristive circuit implementation of a self-repairing network based on biological astrocytes in robot application". In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[117] Mark Horowitz. "1.1 computing's energy problem (and what we can do about it)". In: *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. 2014, pp. 10–14.

[118] Miao Hu, Hai Li, Yiran Chen, Xiaobin Wang, and Robinson E Pino. "Geometry variations analysis of TiO 2 thin-film and spintronic memristors". In: *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*. 2011, pp. 25–30.

[119] Kejie Huang, Rong Zhao, Wei He, and Yong Lian. "High-density and high-reliability nonvolatile field-programmable gate array with stacked 1D2R RRAM array". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.1 (2015), pp. 139–150.

[120] Wenqin Huangfu, Lixue Xia, Ming Cheng, Xiling Yin, Tianqi Tang, Boxun Li, Krishnendu Chakrabarty, Yuan Xie, Yu Wang, and Huazhong Yang. "Computation-oriented fault-tolerance schemes for RRAM computing systems". In: *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2017, pp. 794–799.

[121] Brian R Hunt, Judy A Kennedy, Tien-Yien Li, and Helena E Nusse. *The theory of chaotic attractors*. Springer Science & Business Media, 2013.

[122] Daniele Ielmini and H-S Philip Wong. "In-memory computing with resistive switching devices". In: *Nature electronics* 1.6 (2018), pp. 333–343.

[123] Giacomo Indiveri. "A low-power adaptive integrate-and-fire neuron circuit". In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 4. 2003, pp. IV–IV.

[124] Giacomo Indiveri, Elisabetta Chicca, and Rodney Douglas. "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity". In: *IEEE Transactions on Neural Networks and Learning Systems* 17.1 (2006), pp. 211–221.

[125] Muhammad Irfan, Qaiser Habib, Ghulam M Hassan, Khawaja M Yahya, and Samira Hayat. "Combinational digital circuit synthesis using Cartesian Genetic Programming from a NAND gate template". In: *The 6th International Conference on Emerging Technologies (ICET)*. 2010, pp. 343–347.

[126] Masaya Iwata, Isamu Kajitani, Hitoshi Yamada, Hitoshi Iba, and Tetsuya Higuchi. "A pattern recognition system using evolvable hardware". In: *Parallel Problem Solving from Nature (PPSN)*. 1996, pp. 761–770.

[127] Herbert Jaeger. "Echo state network". In: *Scholarpedia* 2.9 (2007), p. 2330.

[128] Herbert Jaeger. "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note". In: *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report* 148.34 (2001), p. 13.

[129] Herbert Jaeger and Harald Haas. "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication". In: *Science* 304.5667 (2004), pp. 78–80.

[130] Herbert Jaeger, Wolfgang Maass, and Jose Principe. "Special issue on echo state networks and liquid state machines." In: *Neural Network* 20.3 (2007), pp. 287–289.

[131] David Jefferson, Robert Collins, Claus Cooper, Michael Dyer, Margot Flowers, Richard Korf, Charles Taylor, and Alan Wang. *Evolution as a theme in artificial life: The genesys/tracker system*. 1990.

[132] Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J Spanos. "Towards efficient data valuation based on the shapley value". In: *The 22nd International Conference on Artificial Intelligence and Statistics (ICAIAS)*. 2019, pp. 1167–1176.

[133] Yogesh N Joglekar and Stephen J Wolf. "The elusive memristor: properties of basic electrical circuits". In: *European Journal of Physics* 30.4 (2009), p. 661.

[134] Pilin Junsangsri and Fabrizio Lombardi. "Design of a hybrid memory cell using memristance and ambipolarity". In: *IEEE Transactions on Nanotechnology* 12.1 (2012), pp. 71–80.

[135] Andrew B Kahng and Sudhakar Muddu. "An analytical delay model for RLC interconnects". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16.12 (1997), pp. 1507–1514.

[136] Isamu Kajitani, Tsutomu Hoshino, Daisuke Nishikawa, Hiroshi Yokoi, Shougo Nakaya, Tsukasa Yamauchi, Takeshi Inuo, Nobuki Kajihara, Masaya Iwata, Didier Keymeulen, et al. "A gate-level EHW chip: Implementing GA operations and reconfigurable hardware on a single LSI". In: *International Conference on Evolvable Systems (ICS)*. Springer. 1998, pp. 1–12.

[137] Tatiana Kalganova. "An Extrinsic Function-Level Evolvable Hardware Approach". In: *European Conference on Genetic Programming (ECGP)*. 2000.

[138] Tatiana Kalganova and Julian Miller. "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness". In: *Proceedings of the 1st NASA/DoD Workshop on Evolvable Hardware (NEHW)*. 1999, pp. 54–63.

[139] Lars Keuninckx, Jan Danckaert, and Guy Van der Sande. "Real-time audio processing with a cascade of discrete-time delay line-based reservoir computers". In: *Cognitive Computation* 9.3 (2017), pp. 315–326.

[140] Didier Keymeulen, Ricardo Salem Zebulum, Yili Jin, and Adrian Stoica. "Fault-tolerant evolvable hardware using field-programmable transistor arrays". In: *IEEE Transactions on Reliability* 49.3 (2000), pp. 305–316.

[141] Hyongsuk Kim, Maheshwar Pd Sah, Changju Yang, Tamás Roska, and Leon O Chua. "Memristor bridge synapses". In: *Proceedings of IEEE* 100.6 (2011), pp. 2061–2070.

[142] Kuk-Hwan Kim, Siddharth Gaba, Dana Wheeler, Jose M Cruz-Albrecht, Tahir Hussain, Narayan Srinivasa, and Wei Lu. "A functional hybrid memristor crossbar-array/CMOS system for data storage and neuromorphic applications". In: *Nano Letters* 12.1 (2012), pp. 389–395.

[143] Seungjun Kim, Hu Young Jeong, Sung Kyu Kim, Sung-Yool Choi, and Keon Jae Lee. "Flexible memristive memory array on plastic substrates". In: *Nano Letters* 11.12 (2011), pp. 5438–5442.

[144] Tae-Hyun Kim, Byungkyu Song, In-Jun Jung, and Seong-Ook Jung. "A Sneak Current Compensation Scheme With Offset Cancellation Sensing Circuit for ReRAM-Based Cross-Point Memory Array". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 69.4 (2021), pp. 1583–1594.

[145] Denis Kleyko, Edward Paxon Frady, Mansour Kheffache, and Evgeny Osipov. "Integer echo state networks: efficient reservoir computing for digital hardware". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.4 (2020), pp. 1688–1701.

[146] Eric AM Klumperink, Federico Bruccoleri, and Bram Nauta. "Finding all elementary circuits exploiting transconductance". In: *IEEE Transactions on Circuits and Systems II: Express Briefs - Analog and Digital Signal Processing* 48.11 (2001), pp. 1039–1053.

[147] Felix Köster, Dominik Ehlert, and Kathy Lüdge. "Limitations of the Recall Capabilities in Delay-Based Reservoir Computing Systems". In: *Cognitive Computation* (2020), pp. 1–8.

[148] John R Koza et al. *Genetic programming II*. Vol. 17. MIT press Cambridge, 1994.

[149] John R Koza, David Andre, Forrest H Bennett III, and Martin A Keane. "Use of automatically defined functions and architecture-altering operations in automated circuit synthesis with genetic programming". In: *Proceedings of 1st Annual Conference on Genetic Programming (ACGP)*. 1996, pp. 132–140.

[150] John R Koza, David Andre, Martin A Keane, and Forrest H Bennett III. *Genetic programming III: Darwinian invention and problem solving*. Vol. 3. Morgan Kaufmann, 1999.

[151] John R Koza, Forrest H Bennett, David Andre, and Martin A Keane. "Automated design of both the topology and sizing of analog electrical circuits using genetic programming". In: *Artificial Intelligence in Design'96*. Springer, 1996, pp. 151–170.

[152] John R. Koza, Forrest H Bennett, David Andre, Martin A. Keane, and Frank Dunlap. "Automated synthesis of analog electrical circuits by means of genetic programming". In: *IEEE Transactions on Evolutionary Computation* 1.2 (1997), pp. 109–128.

[153] Olga Krestinskaya, Alex Pappachen James, and Leon Ong Chua. "Neuromemristive circuits for edge computing: A review". In: *IEEE Transactions on Neural Networks and Learning Systems* (2019).

[154] R Krohling, Yuchao Zhou, A Tyrrell, et al. "Evolving FPGA-based robot controllers using an evolutionary algorithm". In: *1st International Conference on Artificial Immune Systems (ICARIS)*. 2002.

[155] Wim Kruiskamp and Domine Leenaerts. "DARWIN: CMOS opamp synthesis by means of a genetic algorithm". In: *Proceedings of the 32nd Annual ACM/IEEE Design Automation Conference (DAC)*. 1995, pp. 433–438.

[156] Manjari S Kulkarni and Christof Teuscher. "Memristor-based reservoir computing". In: *Proceedings of the IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*. 2012, pp. 226–232.

[157] T Nandha Kumar, Haider AF Almurib, and Fabrizio Lombardi. "A novel design of a memristor-based look-up table (LUT) for FPGA". In: *2014 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. 2014, pp. 703–706.

[158] Yuki Kume, Song Bian, and Takashi Sato. "A Tuning-Free Hardware Reservoir Based on MOSFET Crossbar Array for Practical Echo State Network Implementation". In: *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2020, pp. 458–463.

[159] Shahar Kvatinsky, Eby G Friedman, Avinoam Kolodny, and Uri C Weiser. "TEAM: Threshold adaptive memristor model". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 60.1 (2012), pp. 211–221.

[160] Shahar Kvatinsky, Misbah Ramadan, Eby G Friedman, and Avinoam Kolodny. "VTEAM: A general model for voltage-controlled memristors". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 62.8 (2015), pp. 786–790.

[161] Cyrille Lambert, Tatiana Kalganova, and Emanuele Stomeo. "FPGA-based systems for evolvable hardware". In: *Conference of the World Academy of Science, Engineering and Technology*. 2006, pp. 123–129.

[162] Jörg Langeheine, Joachim Becker, Simon Folling, Karlheinz Meier, and Johannes Schemmel. "A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits". In: *Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001*. IEEE. 2001, pp. 172–175.

[163] Kenneth Levenberg. "A method for the solution of certain non-linear problems in least squares". In: *Quarterly of applied mathematics* 2.2 (1944), pp. 164–168.

[164] Omer Levy, Kenton Lee, Nicholas FitzGerald, and Luke Zettlemoyer. "Long short-term memory as a dynamically computed element-wise weighted sum". In: *arXiv preprint arXiv:1805.03716* (2018).

[165] Bo Li and Guoyong Shi. "A Native SPICE Implementation of Memristor Models for Simulation of Neuromorphic Analog Signal Processing Circuits". In: *ACM Transactions on Design Automation of Electronic Systems* 27.1 (2021), pp. 1–24.

[166] Huihan Li, Shaocong Wang, Xumeng Zhang, Wei Wang, Rui Yang, Zhong Sun, Wanxiang Feng, Peng Lin, Zhongrui Wang, Linfeng Sun, et al. "Memristive crossbar arrays for storage and computing applications". In: *Advanced Intelligent Systems* 3.9 (2021), p. 2100017.

[167] Jialing Li, Kangjun Bai, Lingjia Liu, and Yang Yi. "A deep learning based approach for analog hardware implementation of delayed feedback reservoir computing system". In: *2018 19th International Symposium on Quality Electronic Design (ISQED)*. 2018, pp. 308–313.

[168] Tianjian Li, Xiangyu Bi, Naifeng Jing, Xiaoyao Liang, and Li Jiang. "Sneak-path based test and diagnosis for 1R RRAM crossbar using voltage bias technique". In: *Proceedings of the 54th Annual Design Automation Conference (DAC)*. 2017, pp. 1–6.

[169] Yesheng Li and Kah-Wee Ang. "Hardware implementation of neuromorphic computing using large-scale memristor crossbar arrays". In: *Advanced Intelligent Systems* 3.1 (2021), p. 2000137.

[170] Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J Joshua Yang. "Review of memristor devices in neuromorphic computing: materials sciences and device challenges". In: *Journal of Physics D: Applied Physics* 51.50 (2018), p. 503002.

[171] Sam Lilak, Walt Woods, Kelsey Scharnhorst, Christopher Dunham, Christof Teuscher, Adam Z Stieg, and James K Gimzewski. "Spoken digit classification by in-materio reservoir computing with neuromorphic atomic switch networks". In: *Frontiers in Nanotechnology* (2021), p. 38.

[172] Jilan Lin, Cheng-Da Wen, Xing Hu, Tianqi Tang, Chao Lin, Yu Wang, and Yuan Xie. "Rescuing rram-based computing from static and dynamic faults". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40.10 (2020), pp. 2049–2062.

[173] Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. "Learning long-term dependencies in NARX recurrent neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 7.6 (1996), pp. 1329–1338.

[174] Bo Liu, Yan Wang, Zhiping Yu, Leibo Liu, Miao Li, Zheng Wang, Jing Lu, and Francisco V Fernández. "Analog circuit optimization system based on hybrid evolutionary algorithms". In: *Integration* 42.2 (2009), pp. 137–148.

[175] Leibo Liu, Jianfeng Zhu, Zhaoshi Li, Yanan Lu, Yangdong Deng, Jie Han, Shouyi Yin, and Shaojun Wei. "A Survey of Coarse-Grained Reconfigurable Architecture and Design: Taxonomy, Challenges, and Applications". In: *ACM Computing Surveys* 52.6 (2019), pp. 1–39.

[176] Weixin Liu, Masahiro Murakawa, and Tetsuya Higuchi. "ATM cell scheduling by function level evolvable hardware". In: *International Conference on Evolvable Systems*. Springer. 1996, pp. 180–192.

[177] Alon Loeffler, Ruomin Zhu, Joel Hochstetter, Adrian Diaz-Alvarez, Tomonobu Nakayama, James M Shine, and Zdenka Kuncic. "Modularity and multitasking in neuro-memristive reservoir networks". In: *Neuromorphic Computing and Engineering* 1.1 (2021), p. 014003.

[178] Jason D Lohn and Silvano P Colombano. "A circuit representation technique for automated circuit design". In: *IEEE Transactions on Evolutionary Computation* 3.3 (1999), pp. 205–219.

[179] Lisa Loomis, Nathan McDonald, and Cory Merkel. "An FPGA implementation of a time delay reservoir using stochastic logic". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 14.4 (2018), pp. 1–15.

[180] Mantas Lukoševičius and Herbert Jaeger. "Reservoir computing approaches to recurrent neural network training". In: *Computer Science Review* 3.3 (2009), pp. 127–149.

[181] Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*. 2017, pp. 4768–4777.

[182] Qianli Ma, Lifeng Shen, and Garrison W Cottrell. "DeePr-ESN: A deep projection-encoding echo-state network". In: *Information Sciences* 511 (2020), pp. 152–171.

[183] Wolfgang Maass and Henry Markram. "On the computational power of circuits of spiking neurons". In: *Journal of Computer and System Sciences* 69.4 (2004), pp. 593–616.

[184] Wolfgang Maass, Thomas Natschläger, and Henry Markram. "Real-time computing without stable states: A new framework for neural computation based on perturbations". In: *Neural Computation* 14.11 (2002), pp. 2531–2560.

[185] Michael C Mackey and Leon Glass. "Oscillation and chaos in physiological control systems". In: *Science* 197.4300 (1977), pp. 287–289.

[186] Abdul Manazir and Khalid Raza. "Recent developments in Cartesian genetic programming and its variants". In: *ACM Computing Surveys* 51.6 (2019), pp. 1–29.

[187] Harika Manem, Garrett S Rose, Xiaoli He, and Wei Wang. "Design considerations for variation tolerant multilevel CMOS/Nano memristor memory". In: *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*. 2010, pp. 287–292.

[188] Tomáš Martínek and Lukáš Sekanina. "An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga". In: *Evolvable Systems: From Biology to Hardware: 6th International Conference, (ICES)*. Springer. 2005, pp. 76–85.

[189] Claudio Mattiussi. *Evolutionary synthesis of analog networks*. Tech. rep. EPFL, 2005.

[190] Claudio Mattiussi and Dario Floreano. "Analog genetic encoding for the evolution of circuits and networks". In: *IEEE Transactions on Evolutionary Computation* 11.5 (2007), pp. 596–607.

[191] Gerard CM Meijer. "Thermal sensors based on transistors". In: *Sensors and Actuators* 10.1-2 (1986), pp. 103–125.

[192] Cory Merkel. "Design of a time delay reservoir using stochastic logic: A feasibility study". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2186–2192.

[193] Cory Merkel, Qutaiba Saleh, Colin Donahue, and Dhireesha Kudithipudi. "Memristive reservoir computing architecture for epileptic seizure detection". In: *Procedia Computer Science* 41 (2014), pp. 249–254.

[194] A Mesquita, Fabio A Salazar, and P Paulo Canazio. "Chromosome representation through adjacency matrix in evolutionary circuits synthesis". In: *Proceedings of 2002 NASA/DoD Conference on Evolvable Hardware (EH)*. 2002, pp. 102–109.

[195] Carl Dean Meyer. "Matrix analysis and applied linear algebra book and solutions manual". In: *SIAM Journal on Mathematical Analysis* (2000).

[196] Julian F Miller, Tatiana Kalganova, Dominic Job, and Natalia Lipnitskaya. "The genetic algorithm as a discovery engine: Strange circuits and new principles". In: *Creative Evolutionary Systems*. Elsevier, 2002, pp. 443–466.

[197] Julian F Miller and Stephen L Smith. "Redundancy and computational efficiency in cartesian genetic programming". In: *IEEE Transactions on Evolutionary Computation* 10.2 (2006), pp. 167–174.

[198] Julian F Miller and Peter Thomson. "Aspects of digital evolution: Evolvability and architecture". In: *International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer. 1998, pp. 927–936.

[199] Julian F Miller, Peter Thomson, and Terence Fogarty. *Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study*. 1997.

[200] Christine M Miranda and Allen C Keeney. "Manufacturing methodology enhancements for space systems hardware fabrication". In: *2017 IEEE Aerospace Conference*. 2017, pp. 1–11.

[201] Oliver Mitea, Markus Meissner, Lars Hedrich, and Peter Jores. "Automated constraint-driven topology synthesis for analog circuits". In: *2011 Design, Automation & Test in Europe (DATE)*. 2011, pp. 1–4.

[202] Junichi Mizoguchi, Hitoshi Hemmi, and Katsunori Shimohara. "Production genetic algorithms for automated hardware design through an evolutionary process". In: *Proceedings of the1stt IEEE Conference on Evolutionary Computation (CEC)*. 1994, pp. 661–664.

[203] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science". In: *Nature Communications* 9.1 (2018), pp. 1–12.

[204] Manobendra Nath Mondal, Susmita Sur-Kolay, and Bhargab B Bhattacharya. "Test Optimization in Memristor Crossbars Based on Path Selection". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42.1 (2022), pp. 294–307.

[205] John Moon, Wen Ma, Jong Hoon Shin, Fuxi Cai, Chao Du, Seung Hwan Lee, and Wei D Lu. "Temporal data classification and forecasting using a memristor-based reservoir computing system". In: *Nature Electronics* 2.10 (2019), pp. 480–487.

[206] Alberto Moraglio and Krzysztof Krawiec. "Geometric semantic genetic programming for recursive boolean programs". In: *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017, pp. 993–1000.

[207] Masahiro Murakawa, Shuji Yoshizawa, and Tetsuya Higuchi. "Adaptive equalization of digital communication channels using evolvable hardware". In: *International Conference on Evolvable Systems*. Springer. 1996, pp. 377–389.

[208] Masahiro Murakawa, Shuji Yoshizawa, Isamu Kajitani, Tatsumi Furuya, Masaya Iwata, and Tetsuya Higuchi. "Hardware evolution at function level". In: *International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer. 1996, pp. 62–71.

[209] Naveen Musunuru et al. "Modeling long range dependence in wheat food price returns". In: *International Journal of Finance & Economics* 11.9 (2019), pp. 1–46.

[210] William Mydlowec and John Koza. "Use of time-domain simulations in automatic synthesis of computational circuits using genetic programming". In: *Genetic and Evolutionary Computation Conference*. 2000, pp. 187–197.

[211] Laurence Nagel and Donald O Pederson. *SPICE (simulation program with integrated circuit emphasis)*. 1973. URL: https://escholarship.org/uc/item/0505x998.

[212] Joma Nakayama, Kazutaka Kanno, and Atsushi Uchida. "Laser dynamical reservoir computing with consistency: an approach of a chaos mask signal". In: *Optics Express* 24.8 (2016), pp. 8679–8692.

[213] Masanori Natsui, Naofumi Homma, Takafumi Aoki, and Tatsuo Higuchi. "Topology-oriented design of analog circuits based on evolutionary graph generation". In: *International Conference on Parallel Problem Solving from Nature (PPSN)*. Springer. 2004, pp. 342–351.

[214] Danko Nikolic, Stefan Haeusler, Wolf Singer, and Wolfgang Maass. "Temporal dynamics of information content carried by neurons in the primary visual cortex". In: *NIPS*. 2006, pp. 1041–1048.

[215] Dimin Niu, Cong Xu, Naveen Muralimanohar, Norman P Jouppi, and Yuan Xie. "Design trade-offs for high density cross-point resistive memory". In: *Proceedings of the 2012 ACM/IEEE international symposium on Low power electronics and design*. 2012, pp. 209–214.

[216] Michael O'Neill and Conor Ryan. "Grammatical evolution". In: *IEEE Transactions on Evolutionary Computation* 5.4 (2001), pp. 349–358.

[217] Michael O'Neill. *Riccardo Poli, William B. Langdon, Nicholas F. McPhee: a field guide to genetic programming*. 2009.

[218] Emil S Ochotta, Rob A Rutenbar, and L Richard Carley. "Synthesis of high-performance analog circuits in ASTRX/OBLX". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15.3 (1996), pp. 273–294.

[219] Rashad S Oreifej, Rawad N Al-Haddad, Heng Tan, and Ronald F DeMara. "Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II Pro devices". In: *International Conference on Field Programmable Logic and Applications (FPL)*. 2007, pp. 299–304.

[220] Ehsan Nedaaee Oskoee and Muhammad Sahimi. "Electric currents in networks of interconnected memristors". In: *Physical Review E* 83.3 (2011), p. 031105.

[221] Yvan Paquot, Francois Duport, Antoneo Smerieri, Joni Dambre, Benjamin Schrauwen, Marc Haelterman, and Serge Massar. "Optoelectronic reservoir computing". In: *Scientific Reports* 2 (2012), p. 287.

[222] Jaeyoung Park. "Neuromorphic computing using emerging synaptic devices: A retrospective summary and an outlook". In: *Electronics* 9.9 (2020), p. 1414.

[223] Alexander G Parlos, Omar T Rais, and Amir F Atiya. "Multi-step-ahead prediction using dynamic recurrent neural networks". In: *Neural Network* 13.7 (2000), pp. 765–786.

[224] Tomasz P Pawlak and Krzysztof Krawiec. "Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis". In: *Evolutionary Computation* 26.2 (2018), pp. 177–212.

[225] Rubén Salvador Perea. "Parametric and structural self-adaptation of embedded systems using evolvable hardware". PhD thesis. Universidad Politécnica de Madrid, 2015.

[226] Shuang Pi, Can Li, Hao Jiang, Weiwei Xia, Huolin Xin, J Joshua Yang, and Qiangfei Xia. "Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension". In: *Nature Nanotechnology* 14.1 (2019), pp. 35–39.

[227] Robinson E Pino and James W Bohl. *Self-reconfigurable memristor-based analog resonant computer*. 2012.

[228] Cosmin Popa. "Low-voltage improved accuracy Gaussian function generator with fourth-order approximation". In: *Microelectronics Journal* 43.8 (2012), pp. 515–520.

[229] Themistoklis Prodromakis, Boon Pin Peh, Christos Papavassiliou, and Christofer Toumazou. "A versatile memristor model with nonlinear dopant kinetics". In: *IEEE Transactions on Electron Devices* 58.9 (2011), pp. 3099–3105.

[230] Paul R Prucnal, Bhavin J Shastri, Thomas Ferreira de Lima, Mitchell A Nahmias, and Alexander N Tait. "Recent progress in semiconductor excitable lasers for photonic spike processing". In: *Advances in Optics and Photonics* 8.2 (2016), pp. 228–299.

[231] Thomas Quarles, AR Newton, DO Pederson, and A Sangiovanni-Vincentelli. *SPICE 3 Version 3F5 User's Manual*. 1994.

[232] Fathalla A Rihan and Bassel F Rihan. "Numerical modelling of biological systems with memory using delay differential equations". In: *Applied Mathematics & Information Sciences* 9.3 (2015), p. 1645.

[233] Ali Rodan and Peter Tino. "Minimum complexity echo state network". In: *IEEE Transactions on Neural Networks and Learning Systems* 22.1 (2010), pp. 131–144.

[234] Ali Rodan and Peter Tiňo. "Simple deterministically constructed cycle reservoirs with regular jumps". In: *Neural Computation* 24.7 (2012), pp. 1822–1852.

[235] Žiga Rojec, Árpád Bűrmen, and Iztok Fajfar. "An evolution-driven analog circuit topology synthesis". In: *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2016, pp. 1–6.

[236] Žiga Rojec, Árpád Bűrmen, and Iztok Fajfar. "Analog circuit topology synthesis by means of evolutionary computation". In: *Engineering Applications of Artificial Intelligence* 80 (2019), pp. 48–65.

[237] Alvin E Roth. *The Shapley value: essays in honor of Lloyd S. Shapley*. Cambridge University Press, 1988.

[238] Subhrajit Roy, Amitava Banerjee, and Arindam Basu. "Liquid state machine with dendritically enhanced readout for low-power, neuromorphic VLSI implementations". In: *IEEE Transactions on Biomedical Circuits and Systems* 8.5 (2014), pp. 681–695.

[239] Popa Rustem. "Genetic Algorithms: An Overview with Applications in Evolvable Hardware". In: *Bio-Inspired Computational Algorithms and Their Applications* (2012), p. 105.

[240] Farzad Sabzikar, A Ian McLeod, and Mark M Meerschaert. "Parameter estimation for ARTFIMA time series". In: *Journal of Statistical Planning and Inference* 200 (2019), pp. 129–145.

[241] Hasim Sak, Andrew W Senior, and Françoise Beaufays. "Long short-term memory recurrent neural network architectures for large scale acoustic modeling". In: *INTERSPEECH* (2014).

[242] Ruben Salvador, Andres Otero, Javier Mora, Eduardo de la Torre, Teresa Riesgo, and Lukas Sekanina. "Self-reconfigurable evolvable hardware system for adaptive image processing". In: *IEEE Transactions on Computers* 62.8 (2013), pp. 1481–1493.

[243] Ruben Salvador, Andres Otero, Javier Mora, Eduardo de la Torre, Lukas Sekanina, and Teresa Riesgo. "Fault tolerance analysis and self-healing strategy of autonomous,

evolvable hardware systems". In: *2011 International Conference on Reconfigurable Computing and FPGAs*. 2011, pp. 164–169.

[244] C.C. Santini, J.F.M. Amaral, M.A.C. Pacheco, M.M. Vellasco, and M.H. Szwarcman. "Evolutionary analog circuit design on a programmable analog multiplexer array". In: *2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings.* 2002, pp. 189–196. DOI: 10.1109/FPT.2002.1188681.

[245] Yerbol A Sapargaliyev and Tatiana G Kalganova. "Open-ended evolution to discover analogue circuits for beyond conventional applications". In: *Genetic Programming and Evolvable Machines* 13.4 (2012), pp. 411–443.

[246] Jürgen Schmidhuber, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez. "Training recurrent networks by evolino". In: *Neural Computation* 19.3 (2007), pp. 757–779.

[247] Benjamin Schrauwen, Michiel D'Haene, David Verstraeten, and Jan Van Campenhout. "Compact hardware liquid state machines on FPGA for real-time speech recognition". In: *Neural Network* 21.2-3 (2008), pp. 511–523.

[248] Benjamin Schrauwen, David Verstraeten, and Jan Van Campenhout. "An overview of reservoir computing: theory, applications and implementations". In: *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN)*. 2007, pp. 471–482.

[249] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. "A survey of neuromorphic computing and neural networks in hardware". In: *arXiv preprint arXiv:1705.06963* (2017).

[250] Adel Sedra and Kenneth Smith. "A second-generation current conveyor and its applications". In: *IEEE Transactions on circuit theory* 17.1 (1970), pp. 132–134.

[251] Lukáš Sekanina. "Evolutionary functional recovery in virtual reconfigurable circuits". In: *ACM Journal on Emerging Technologies in Computing Systems* 3.2 (2007), 8–es.

[252] Lukáš Sekanina. "Virtual reconfigurable circuits for real-world applications of evolvable hardware". In: *International Conference on Evolvable Systems (ICS)*. Springer. 2003, pp. 186–197.

[253] Lukas Sekanina, Vojtech Salajka, and Zdenek Vasicek. "Two-step evolution of polymorphic circuits for image multi-filtering". In: *IEEE Congress on Evolutionary Computation*. 2012.

[254] Lukás Sekanina and Stepan Friedl. "An Evolvable Combinational Unit for FPGAs". In: *Computing & Informatics* 23.5 (2005), pp. 461–486.

[255] Luís F Seoane. "Evolutionary aspects of reservoir computing". In: *Philosophical Transactions of the Royal Society B* 374.1774 (2019), p. 20180377.

[256] Alexantrou Serb, Ali Khiat, and Themistoklis Prodromakis. "Seamlessly fused digital-analogue reconfigurable computing using memristors". In: *Nature Communications* 9.1 (2018), pp. 1–7.

[257] Lloyd S Shapley. *A value for n-person games*. Princeton University Press, 2016.

[258] Xinming Shi, Jiashi Gao, Leandro L Minku, JQ James, and Xin Yao. "Second-order Time Delay Reservoir Computing for Nonlinear Time Series Problems". In: *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. 2021, pp. 1–8.

[259] Xinming Shi, Jiashi Gao, Leandro L Minku, and Xin Yao. "Evolving parsimonious circuits through shapley value-based genetic programming". In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2022, pp. 602–605.

[260] Xinming Shi, Leandro Minku, and Xin Yao. "Evolving Memristive Reservoir". In: *IEEE Transactions on Neural Networks and Learning Systems* (2023). DOI: 10.1109/TNNLS.2023.3270224.

[261] Xinming Shi, Leandro Minku, and Xin Yao. "Harnessing Sneak Currents in 1R Memristor-based Crossbar by Spatiotemporal Evolution for Reservoir Computing". In: *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[262] Xinming Shi, Leandro Minku, and Xin Yao. "Time Delay Reservoir with Enhanced Memory and Its Fully Analog Memristive Circuit". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and System* (2023).

[263] Xinming Shi, Leandro L Minku, and Xin Yao. "A novel tree-based representation for evolving analog circuits and its application to memristor-based pulse generation circuit". In: *Genetic Programming and Evolvable Machines* 23.4 (2022), pp. 453–493.

[264] Xinming Shi, Leandro L Minku, and Xin Yao. "Adaptive Memory-enhanced Time Delay Reservoir and Its Memristive Implementation". In: *IEEE Transactions on Computers* (2022).

[265] Xinming Shi, Zhigang Zeng, Le Yang, and Yi Huang. "Memristor-based circuit design for neuron with homeostatic plasticity". In: *IEEE Transactions on Emerging Topics in Computing Intelligence* 2.5 (2018), pp. 359–370.

[266] Henry O Sillin, Renato Aguilera, Hsien-Hang Shieh, Audrius V Avizienis, Masakazu Aono, Adam Z Stieg, and James K Gimzewski. "A theoretical and experimental study of neuromorphic atomic switch networks for reservoir computing". In: *Nanotechnology* 24.38 (2013), p. 384004.

[267] Sara Silva and Ernesto Costa. "Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories". In: *Genetic Programming and Evolvable Machines* 10.2 (2009), pp. 141–179.

[268] Arpita Sinha, Manjari S Kulkarni, and Christof Teuscher. "Evolving nanoscale associative memories with memristors". In: *Proceeding of the 11th IEEE International Conference on Nanotechnology (NANO)*. 2011, pp. 860–864.

[269] Mark D Skowronski and John G Harris. "Automatic speech recognition using a predictive echo state network classifier". In: *Neural Network* 20.3 (2007), pp. 414–423.

[270] Samin Ebrahim Sorkhabi and Lihong Zhang. "Automated topology synthesis of analog and RF integrated circuits: A survey". In: *Integration* 56 (2017), pp. 128–138.

[271] Nicholas Soures, Lydia Hays, and Dhireesha Kudithipudi. "Robustness of a memristor based liquid state machine". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. 2017, pp. 2414–2420.

[272] Thanwa Sripramong and Christofer Toumazou. "The invention of CMOS amplifiers using genetic programming and current-flow analysis". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21.11 (2002), pp. 1237–1252.

[273] Pushkar Srivastava, Ravindra K Sharma, and Rajeev K Ranjan. "On the realisation of current-mode four-quadrant CMOS cuber". In: *Analog Integrated Circuits and Signal Processing* 99 (2019), pp. 47–61.

[274] R Stanley Williams. "How we found the missing memristor". In: *Chaos, CNN, Memristors and Beyond: A Festschrift for Leon Chua With DVD-ROM, composed by Eleonora Bilotta*. World Scientific, 2013, pp. 483–489.

[275] Spyros Stathopoulos, Ali Khiat, Maria Trapatseli, Simone Cortese, Alexantrou Serb, Ilia Valov, and Themis Prodromakis. "Multibit memory operation of metal-oxide bilayer memristors". In: *Scientific Reports* 7.1 (2017), pp. 1–7.

[276] Jochen J. Steil. "Online reservoir adaptation by intrinsic plasticity for backpropagation-decorrelation and echo state learning." In: *Neural Networks* 20.3 (2007), pp. 353–364.

[277] Thomas Stepleton, Razvan Pascanu, Will Dabney, Siddhant M Jayakumar, Hubert Soyer, and Remi Munos. "Low-pass Recurrent Neural Networks-A memory archi-

tecture for longer-term correlation discovery". In: *arXiv preprint arXiv:1805.04955* (2018).

[278] Julian Stier, Gabriele Gianini, Michael Granitzer, and Konstantin Ziegler. "Analysing neural network topologies: a game theoretic approach". In: *Procedia Computer Science* 126 (2018), pp. 234–243.

[279] Emanuele Stomeo, Tatiana Kalganova, and Cyrille Lambert. "Generalized disjunction decomposition for evolvable hardware". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 36.5 (2006), pp. 1024–1043.

[280] Matthew J Streeter, Martin A Keane, and John R Koza. "Iterative refinement of computational circuits using genetic programming". In: *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. 2002, pp. 877–884.

[281] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. "The missing memristor found". In: *Nature* 453.7191 (2008), pp. 80–83.

[282] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. "The missing memristor found". In: *Nature* 453.7191 (2008), pp. 80–83.

[283] Bidyadhar Subudhi and Debashisha Jena. "A differential evolution based neural network approach to nonlinear system identification". In: *Applied Soft Computing* 11.1 (2011), pp. 861–871.

[284] G Sussman and Richard Stallman. "Heuristic techniques in computer-aided circuit analysis". In: *IEEE Transactions on Circuits and Systems* 22.11 (1975), pp. 857–865.

[285] Seiji Takeda, Daiju Nakano, Toshiyuki Yamane, Gouhei Tanaka, Ryosho Nakane, Akira Hirose, and Shigeru Nakagawa. "Photonic reservoir computing based on laser dynamics with external feedback". In: *Neural Information Processing: 23rd International Conference (ICONIP)*. Springer. 2016, pp. 222–230.

[286] Gouhei Tanaka and Ryosho Nakane. "Simulation platform for pattern recognition based on reservoir computing with memristor networks". In: *Scientific Reports* 12 (2022), p. 9868.

[287] Gouhei Tanaka, Ryosho Nakane, Toshiyuki Yamane, Seiji Takeda, Daiju Nakano, Shigeru Nakagawa, and Akira Hirose. "Waveform classification by memristive reservoir computing". In: *Proceedings of the 24th International Conference on Neural Information Processing (ICNIP)*. Springer. 2017, pp. 457–465.

[288] Gouhei Tanaka, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji Takeda, Hidetoshi Numata, Daiju Nakano, and Akira Hirose. "Recent advances in physical reservoir computing: A review". In: *Neural Network* 115 (2019), pp. 100–123.

[289] Jianshi Tang, Fang Yuan, Xinke Shen, Zhongrui Wang, Mingyi Rao, Yuanyuan He, Yuhao Sun, Xinyi Li, Wenbin Zhang, Yijun Li, et al. "Bridging biological and artificial neural networks with emerging neuromorphic devices: fundamentals, progress, and challenges". In: *Advanced Materials* 31.49 (2019), p. 1902761.

[290] Jun-Yao Tao, Zheng-Mao Wu, Dian-Zuo Yue, Xiang-Sheng Tan, Qing-Qing Zeng, and Guang-Qiong Xia. "Performance enhancement of a delay-based Reservoir computing system by using gradient boosting technology". In: *IEEE Access* 8 (2020), pp. 151990–151996.

[291] Yanyun Tao, Lijun Zhang, and Yuzhen Zhang. "A projection-based decomposition for the scalability of evolvable hardware". In: *Soft Computing* 20.6 (2015).

[292] Yanyun Tao, Yuzhen Zhang, Jian Cao, and Yalong Huang. "A module-level three-stage approach to the evolutionary design of sequential logic circuits". In: *Genetic Programming and Evolvable Machines* 14.2 (2013), pp. 191–219.

[293] Yanyun Tao, Yuzhen Zhang, Jian Cao, and Yalong Huang. "A module-level three-stage approach to the evolutionary design of sequential logic circuits". In: *Genetic Programming & Evolvable Machines* 14.2 (2013), pp. 191–219.

[294] Adrian Thompson. "Evolving electronic robot controllers that exploit hardware resources". In: *European Conference on Artificial Life*. Springer. 1995, pp. 640–656.

[295] Adrian Thompson. "On the automatic design of robust electronics through artificial evolution". In: *International Conference on Evolvable Systems (ICS)*. Springer. 1998, pp. 13–24.

[296] Adrian Thompson et al. "Silicon evolution". In: *Genetic programming* (1996), pp. 444–452.

[297] Jim Torresen. "A divide-and-conquer approach to Evolvable Hardware". In: *Evolvable Systems: From Biology to Hardware*. Ed. by Moshe Sipper, Daniel Mange, and Andrés Pérez-Uribe. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 57–65.

[298] Jim Torresen. "An evolvable hardware tutorial". In: *International Conference on Field Programmable Logic and Applications (FPL)*. Springer. 2004, pp. 821–830.

[299] Antonio C Torrezan, John Paul Strachan, Gilberto Medeiros-Ribeiro, and R Stanley Williams. "Sub-nanosecond switching of a tantalum oxide memristor". In: *Nanotechnology* 22.48 (2011), p. 485203.

[300] Martin Trefzer, Jörg Langeheine, Johannes Schemmel, and Karlheinz Meier. "New genetic operators to facilitate understanding of evolved transistor circuits". In: *Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware (EH)*. 2004, pp. 217–224.

[301] Martin A Trefzer. "Memristor in a Nutshell". In: *Guide to Unconventional Computing for Music* (2017), pp. 159–180.

[302] Martin A Trefzer and Andy M Tyrrell. "Evolvable Hardware". In: *From Practice to Application. Springer* (2015).

[303] Martin Albrecht Trefzer. "Evolution of transistor circuits". PhD thesis. 2006.

[304] Allan Edward Van den Berg and Farouk Smith. "Hardware evolution of a digital circuit using a custom VLSI architecture". In: *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*. 2013, pp. 378–387.

[305] Alvaro Velasquez and Sumit Kumar Jha. "In-memory computing using paths-based logic and heterogeneous components". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 2018, pp. 1512–1515.

[306] David Verstraeten, Joni Dambre, Xavier Dutoit, and Benjamin Schrauwen. "Memory versus non-linearity in reservoirs". In: *The 2010 international joint conference on neural networks (IJCNN)*. 2010, pp. 1–8.

[307] Pietro Verzelli, Cesare Alippi, Lorenzo Livi, and Peter Tiňo. "Input-to-state representation in linear reservoirs dynamics". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[308] Knut Arne Vinger and Jim Torresen. "Implementing evolution of FIR-filters efficiently in an FPGA". In: *Proceedings of NASA/DoD Conference on Evolvable Hardware (EH)*. 2003, pp. 26–29.

[309] Pantelis R Vlachas, Jaideep Pathak, Brian R Hunt, Themistoklis P Sapsis, Michelle Girvan, Edward Ott, and Petros Koumoutsakos. "Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics". In: *Neural Networks* 126 (2020), pp. 191–217.

[310] Holger Vogt, Marcel Hendrix, and Paolo Nenzi. "Ngspice User's Manual Version 31 (Describes ngspice release version)". In: (2019).

[311] Ioannis Vourkas, Athanasios Batsos, and Georgios Ch Sirakoulis. "SPICE modeling of nonlinear memristive behavior". In: *International Journal of Circuit Theory and Applications* 43.5 (2015), pp. 553–565.

[312] Jacques Wainer and Gavin Cawley. "Nested cross-validation when selecting classifiers is overzealous for most practical applications". In: *Expert Systems with Applications* 182 (2021), p. 115222.

[313] James Alfred Walker and Julian Francis Miller. "Investigating the Performance of Module Acquisition in Cartesian Genetic Programming". In: GECCO '05. Association for Computing Machinery, 2005, pp. 1649–1656.

[314] Hsin Pei Wang, Chia Chun Lin, Chia Cheng Wu, Yung Chih Chen, and Chun Yao Wang. "On synthesizing memristor-based logic circuits with minimal operational pulses". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.12 (2018), pp. 2842–2852.

[315] Jin Wang, Chang Hao Piao, and Chong Ho Lee. "Implementing multi-VRC cores to evolve combinational logic circuits in parallel". In: *Evolvable Systems: From Biology to Hardware: 7th International Conference (ICES)*. Springer. 2007, pp. 23–34.

[316] Lin Wang, Huanling Hu, Xue-Yi Ai, and Hua Liu. "Effective electricity energy consumption forecasting using echo state network improved by differential evolution algorithm". In: *Energy* 153 (2018), pp. 801–815.

[317] Qian Wang, Youjie Li, and Peng Li. "Liquid state machine based pattern recognition on FPGA with firing-activity dependent power gating and approximate computing". In: *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*. 2016, pp. 361–364.

[318] Xiaobin Wang, Yiran Chen, Haiwen Xi, Hai Li, and Dimitar Dimitrov. "Spintronic memristor through spin-torque-induced magnetization motion". In: *IEEE Electron Device Letters* 30.3 (2009), pp. 294–297.

[319] Xinjie Wang, Yaochu Jin, and Kuangrong Hao. "Evolving local plasticity rules for synergistic learning in echo state networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.4 (2019), pp. 1363–1374.

[320] Zhongrui Wang, Can Li, Wenhao Song, Mingyi Rao, Daniel Belkin, Yunning Li, Peng Yan, Hao Jiang, Peng Lin, Miao Hu, et al. "Reinforcement learning with analogue memristor arrays". In: *Nature Electronics* 2.3 (2019), pp. 115–124.

[321] Zilu Wang, Xinming Shi, and Xin Yao. "A Brain-Inspired Hardware Architecture for Evolutionary Algorithms Based on Memristive Arrays". In: *ACM Transactions on Design Automation of Electronic Systems* (2023). ISSN: 1084-4309.

[322] Zilu Wang and Xiaoping Wang. "A novel memristor-based circuit implementation of full-function Pavlov associative memory accorded with biological feature". In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.7 (2017), pp. 2210–2220.

[323] Zilu Wang, Xiaoping Wang, Zezao Lu, Weiguo Wu, and Zhigang Zeng. "The design of memristive circuit for affective multi-associative learning". In: *IEEE Transactions on Biomedical Circuits and Systems* 14.2 (2020), pp. 173–185.

[324] Rainer Waser and Masakazu Aono. "Nanoionics-based resistive switching memories". In: *Nanoscience And Technology: A Collection of Reviews from Nature Journals*. World Scientific, 2010, pp. 158–165.

[325] Andreas S Weigend. *Time series prediction: forecasting the future and understanding the past*. Routledge, 2018.

[326] Shiping Wen, Rui Hu, Yin Yang, Tingwen Huang, Zhigang Zeng, and Yong-Duan Song. "Memristor-based echo state network with online least mean square". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49.9 (2018), pp. 1787–1796.

[327] Neil HE Weste and David Harris. *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.

[328] Jayawan HB Wijekoon and Piotr Dudek. "Compact silicon neuron circuit with spiking and bursting behaviour". In: *Neural Networks* 21.2-3 (2008), pp. 524–534.

[329] SA Wolf, DD Awschalom, RA Buhrman, JM Daughton, von S von Molnár, ML Roukes, A Yu Chtchelkanova, and DM Treger. "Spintronics: a spin-based electronics vision for the future". In: *science* 294.5546 (2001), pp. 1488–1495.

[330] Chaoxing Wu, Tae Whan Kim, Hwan Young Choi, Dmitri B Strukov, and J Joshua Yang. "Flexible three-dimensional artificial synapse networks with correlated learning and trainable memory capability". In: *Nature Communications* 8.1 (2017), pp. 1–9.

[331] Ting Wu and Jingsong He. "Random group associated with differential evolution improves scalability of analog circuit design". In: *Sixth International Conference on Advanced Computational Intelligence*. 2013.

[332] Lixue Xia, Peng Gu, Boxun Li, Tianqi Tang, Xiling Yin, Wenqin Huangfu, Shimeng Yu, Yu Cao, Yu Wang, and Huazhong Yang. "Technological exploration of RRAM crossbar array for matrix-vector multiplication". In: *Journal of Computer Science and Technology* 31.1 (2016), pp. 3–19.

[333] Qiangfei Xia and J Joshua Yang. "Memristive crossbar arrays for brain-inspired computing". In: *Nature Materials* 18.4 (2019), pp. 309–323.

[334] Qiangfei Xia, J Joshua Yang, Wei Wu, Xuema Li, and R Stanley Williams. "Self-aligned memristor cross-point arrays fabricated with one nanoimprint lithography step". In: *Nano letters* 10.8 (2010), pp. 2909–2914.

[335] Cong Xu, Xiangyu Dong, Norman P Jouppi, and Yuan Xie. "Design implications of memristor-based RRAM cross-point structures". In: *2011 Design, Automation & Test in Europe (DATE)*. 2011, pp. 1–6.

[336] Qi Xu, Song Chen, Hao Geng, Bo Yuan, Bei Yu, Feng Wu, and Zhengfeng Huang. "Fault tolerance in memristive crossbar-based neuromorphic computing systems". In: *Integration* 70 (2020), pp. 70–79.

[337] Le Yang, Zhigang Zeng, and Yi Huang. "An associative-memory-based reconfigurable memristive neuromorphic system with synchronous weight training". In: *IEEE Transactions on Cognitive and Developmental Systems* 12.3 (2019), pp. 529–540.

[338] Le Yang, Zhigang Zeng, and Xinming Shi. "A memristor-based neural network circuit with synchronous weight adjustment". In: *Neurocomputing* 363 (2019), pp. 114–124.

[339] Xiao Yang, Wanlong Chen, and Frank Z Wang. "Investigations of the staircase memristor model and applications of memristor-based local connections". In: *Analog Integrated Circuits and Signal Processing* 87.2 (2016), pp. 263–273.

[340] Zhenyu Yang, Ke Tang, and Xin Yao. "Large scale evolutionary optimization using cooperative coevolution". In: *Information Sciences* 178.15 (2008), pp. 2985–2999.

[341] Zhenyu Yang, Ke Tang, and Xin Yao. "Scalability of generalized adaptive differential evolution for large-scale continuous optimization". In: *Soft Computing* 15 (2011), pp. 2141–2155.

[342] Xin Yao. "Evolving artificial neural networks". In: *Proceedings of the IEEE* 87.9 (1999), pp. 1423–1447.

[343] Xin Yao and Tetsuya Higuchi. "Promises and challenges of evolvable hardware". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 29.1 (1999), pp. 87–97.

[344] Yang Yi, Yongbo Liao, Bin Wang, Xin Fu, Fangyang Shen, Hongyan Hou, and Lingjia Liu. "FPGA based spike-time dependent encoder and reservoir design in neuromorphic computing processors". In: *Microprocessors and Microsystems: Embedded Hardware Design* 46 (2016), pp. 175–183.

[345] Shimeng Yu. "Neuro-inspired computing with emerging nonvolatile memorys". In: *Proceedings of the IEEE* 106.2 (2018), pp. 260–285.

[346] Tijn van der Zant, Vlatko Bečanović, Kazuo Ishii, Hans-Ulrich Kobialka, and Paul Ploeger. "Finding good echo state networks to control an underwater robot using evolutionary computations". In: *IFAC Proceedings Volumes* 37.8 (2004), pp. 215–220.

[347] Ricardo Salem Zebulum, Marco Aurélio Pacheco, and Marley Maria Be Vellasco. *Evolutionary electronics: automatic design of electronic circuits and systems by genetic algorithms*. CRC press, 2018.

[348] JJ Zhang, HJ Sun, Y Li, Q Wang, XH Xu, and XS Miao. "AgInSbTe memristor with gradual resistance tuning". In: *Applied Physics Letters* 102.18 (2013), p. 183513.

[349] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Russ R Salakhutdinov, and Yoshua Bengio. "Architectural complexity measures of recurrent neural networks". In: *Advances in Neural Information Processing Systems (NIPS)* 29 (2016), pp. 1822–1830.

[350] Wenbin Zhang, Peng Yao, Bin Gao, Qi Liu, Dong Wu, Qingtian Zhang, Yuankun Li, Qi Qin, Jiaming Li, Zhenhua Zhu, et al. "Edge learning using a fully integrated neuro-inspired memristor chip". In: *Science* 381.6663 (2023), pp. 1205–1211.

[351] Yang Zhang, Yi Li, Xiaoping Wang, and Eby G. Friedman. "Synaptic Characteristics of Ag/AgInSbTe/Ta-Based Memristor for Pattern Recognition Applications". In: *IEEE Transactions on Electron Devices* PP.4 (2017), pp. 1–6.

[352] Yide Zhang, Zhigang Zeng, and Shiping Wen. "Implementation of memristive neural networks with spike-rate-dependent plasticity synapses". In: *2014 International Joint Conference on Neural Networks (IJCNN)*. 2014, pp. 2226–2233.

[353] Yong Zhang, Peng Li, Yingyezhe Jin, and Yoonsuck Choe. "A digital liquid state machine with biologically inspired learning and its application to speech recognition". In: *IEEE Transactions on Neural Networks and Learning Systems* 26.11 (2015), pp. 2635–2649.

[354] Jingyu Zhao, Feiqing Huang, Jia Lv, Yanjie Duan, Zhen Qin, Guodong Li, and Guangjian Tian. "Do rnn and lstm have long memory?" In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*. 2020, pp. 11365–11375.

[355] Liang Zhao, Qinghui Hong, and Xiaoping Wang. "Novel designs of spiking neuron circuit and STDP learning circuit based on memristor". In: *Neurocomputing* 314 (2018), pp. 207–214.

[356] Jian Zheng, Zhigang Zeng, and Yidong Zhu. "Memristor-based nonvolatile synchronous flip-flop circuits". In: *2017 Seventh International Conference on Information Science and Technology (ICIST)*. 2017, pp. 504–508.

[357] Yanan Zhong, Jianshi Tang, Xinyi Li, Bin Gao, He Qian, and Huaqiang Wu. "Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing". In: *Nature Communications* 12.1 (2021), pp. 1–9.

[358] Yanan Zhong, Jianshi Tang, Xinyi Li, Xiangpeng Liang, Zhengwu Liu, Yijun Li, Peng Yao, Zhenqi Hao, Bin Gao, He Qian, et al. "Memristor-based fully analog reservoir computing system for power-efficient real-time spatiotemporal signal processing". In: *Nature Electronics* 5 (2022), pp. 672–681.

[359] Xiaojian Zhu, Qiwen Wang, and Wei D Lu. "Memristor networks for real-time neural activity analysis". In: *Nature Communications* 11.1 (2020), pp. 1–9.

[360] Ye Zhuo, Rivu Midya, Wenhao Song, Zhongrui Wang, Shiva Asapu, Mingyi Rao, Peng Lin, Hao Jiang, Qiangfei Xia, R Stanley Williams, et al. "A dynamical compact model of diffusive and drift memristors for neuromorphic computing". In: *Advanced Electronic Materials* 8.8 (2022), p. 2100696.

[361] Mohammed Affan Zidan, Hossam Aly Hassan Fahmy, Muhammad Mustafa Hussain, and Khaled Nabil Salama. "Memristor-based memory: The sneak paths problem and solutions". In: *Microelectronics Journal* 44.2 (2013), pp. 176–183.

[362] Fatima Tuz Zohora, Sutapa Debnath, and ABM Harun-ur Rashid. "Memristor-CMOS hybrid implementation of leaky integrate and fire neuron model". In: *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*. 2019, pp. 1–5.