Adaptive memory-enhanced time delay reservoir and its memristive implementation

Shi, Xinming; Minku, Leandro; Yao, Xin

DOI:
10.1109/TC.2022.3173151

Adaptive Memory-enhanced Time Delay Reservoir and Its Memristive Implementation

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

Abstract—Time Delay Reservoir (TDR) is a hardware-friendly machine learning approach from two perspectives. First, it can prevent the connection overhead of neural networks with increasing neurons. Second, through its dynamic system representation, TDR can also be implemented in hardware by different systems. However, it performs poorly on tasks that involve long-term dependency. In this work, we first introduce 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. 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 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.

Index Terms—Time Delay Reservoir, memristor, time series prediction.

1 Introduction

Reservoir computing (RC) was originally proposed to provide solutions for the shortcomings of conventional recurrent neural networks (RNNs), such as computationally expensive weight update [1]. In RC, a recurrent neural network is randomly created and remains unchanged during training, which is called the reservoir. By virtue of its modeling accuracy, modeling capacity, biological plausibility, as well extensibility and parsimony, RC methods have quickly become popular [2], and constitute one of the basic paradigms of RNN modeling.

Some researchers have implemented high dimensional reservoirs by simulating neural networks, such as Echo State Network (ESN) and Liquid State Machine (LSM), where the different nonlinear activation functions (neurons) and their connections are applied to realize the RC features. However, this means that a reservoir with H neurons will have up to H^2 connections, potentially leading to large area and power overhead when implemented in hardware [1]. In fact the reservoirs do not necessarily need to be neural networks [2]. Some researchers have applied differential equation-based dynamic systems to model the high dimensionality of the reservoir instead of neurons [3]. This differential equation-based RC called Time Delay Reservoir (TDR) can prevent the large overhead of neuron-based RCs such as ESN by using time multiplexing resources [1]. Therefore, TDR has friendly features to physical implementations.

The physical implementation of machine learning approaches has attracted increasing attention in diverse fields of research, as they can have a fast speed of data processing and low learning cost [4]. Depending on different types of physical devices, electronic RC [5], spintronic RC [6], and biological RC [7] have been widely studied, where electronic RC have attracted great attention [8], [5], [9]. One way of realizing electronic RC is to implement neuron-based reservoir using neural network hardware or neuromorphic computing techniques, such as FPGA [8] and MOSFET crossbar array [5]. However, these neuron-based reservoirs may incur a large area and power overhead caused by H^2 connections of neurons. Another method of realizing electronic RC is to employ other dynamical systems instead of neural networks. For that, researchers mainly focus on exploring different dynamic systems that have the features of a reservoir to serve as reservoirs [10] [11] [12].

Memristor is a new-type nonlinear electronic component first predicted by Chua, which can vary its resistance according to applied voltage or current [13]. The memristor is resistance changeable, non-volatile, power-efficient, and high-density integration friendly, so that it is very promising in areas like storage, artificial neural networks, and logic computation [14] [15] [16]. Therefore, researchers have made their attempts to design memristive reservoirs constructing dynamic systems directly by these memristors without neuron circuits, taking advantage of the intrinsic nonlinearity and/or volatile effects of the devices [17]. Many computational problems, such as time series pattern recognition, prediction, and generation, have been addressed [18] [19].

Especially, time series modeling is an important topic in machine learning, and it has been well addressed by a variety of recurrent networks [20] and RC approaches [10]. However, modeling long-range dependence remains a key challenge. To alleviate the issue of vanishing gradients in modeling long-range dependence, much effort has been

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

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

spent on proposing networks and variants to overcome vanishing gradients, e.g., LSTM [21], GRU networks [22], different variants of RNN [23] and LSTM [20]. However, these methods may result in large area and power overhead caused by neuron connections especially from the hardware implementation perspective [10] [1]. Compared with these algorithms, 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 (see the Supplemental Material of [10]). However, its ability to tackle long term dependency tasks still needs further exploitation.

To improve TDR's ability to deal with long term dependency tasks and produce a solution that can be implemented in hardware requiring small area and low power consumption, we propose an adaptive memory-enhanced TDR and its memristive implementation. The main contributions of this work are as follows:

- We propose a novel high-order time delay unit for TDR able to accumulate and transfer long history states.
- This proposed high-order time delay unit is optimizable and adjustable. In particular, the order of the time delay units can be optimized, where the states with the long-term history are accumulated and transferred to the current state, being automatically adjusted to different tasks.
- Particle Swarm Optimisation (PSO) is applied to automate the optimization of the order of the time delay units. As a result, our approach not only improves prediction performance on both short memory and long memory datasets over the existing reservoirs, but also over other heavier approaches.
- We introduce a hardware implementation of our proposed model based on two different types of memristor.
- We show that the circuit area and power consumption of our proposed memristive implementation outperform other existing traditional work.

2 Related Work

2.1 Standard Time Delay Reservoir

Delay differential equations (DDEs) can describe real-world systems, where the events are rarely instantaneous. This can be modelled as follows [24]:

$$y'(t) = f(y(t), y(t-\tau_1), y(t-\tau_2), ..., y(t-\tau_d), t), t \geq t_0. \quad (1)$$

where τ_i are the delay terms, and could be constant, or variable as functions of t or even of the state y. Considering these features of RC, 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 [24]:

$$\dot{h}(t) = -h(t) + f(h(t-\tau), x(t)), \quad (2)$$

where $h(t)$ refers to the states of reservoir, $x(t)$ 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 (2) with integration step θ, the reservoir state $h_i(k)$ could be rewritten as:

$$h_i^{(k)} = \frac{1}{1+\theta} h_{i-1}^{(k)} + \frac{\theta}{1+\theta} f(h_i^{(k-1)}, x_i^{(k)}). \quad (3)$$

Considering that there is an error between the output $\hat{y}^{(k)}$ and the target $y^{(k)}$ defined as $\varepsilon^{(k)}$, forming a sequence of independent and identically distributed (i.i.d.) random vectors received over time, and that W_{yh} represents the output weights, the procedure of standard TDR can be formulated as:

$$\begin{cases} y^{(k)} = W_{yh} h_i^{(k)} + \varepsilon^{(k)} \\ h_i^{(k)} = \frac{1}{1+\theta} h_{i-1}^{(k)} + \frac{\theta}{1+\theta} f(h_i^{(k-1)}, x_i^{(k)}). \end{cases} \quad (4)$$

2.2 Long-term Dependency Problem

Many real world applications produce datasets with long memory effects, including language and music, dendrochronology and hydrology, and financial data [25] [26]. Some researchers apply statistical methods to model long memory datasets [27]. However, such statistical models make strong parametric assumptions which are challenging to be determined in real world problems. Moreover, they appear to be quite rigid and inflexible for real-world applications compared to neural networks [20]. Using neural networks to tackle the long-term dependency has been drawing great attention in recent years. Since the issue of vanishing gradients has been found, there is a large number of studies focusing on this issue [21] [28] [29] [30] [31] [20].

Compared with the aforementioned neuron-based neural networks, TDR is more friendly to hardware implementations due to its dynamic system representation [10]. This is important to enable fast speed of data processing and lower learning cost [4]. However, existing TDRs are not prepared for dealing with the long-term dependency. This paper will address this issue.

2.3 Hardware Implementation of Reservoir Computing

Different types of RC algorithms that have been implemented by hardware. For instance, ESN and LSM are two types of RC algorithms based on nonlinear function neuron and spiking neuron, respectively. Both ESN [8] [32] and LSM [33] [34] [35] models have been fully developed in FPGA for data recognition and classification.

Besides neuron-based RC models, the dynamic system-based RC model, TDR, has also been implemented in hardware. The photonic TDR has recently attracted widespread attention. However, it requires expensive peripheral devices such as a digitizer and waveform generator [11]. Electronic TDR has also been actively studied for developing machine learning devices with low training cost [4]. Some of the electronic TDRs are mostly built on traditional CMOS devices combined with other components such as capacitor and operational amplifier [12] [10] [11]. There have been several TDRs implemented based on the emerging electronic device named memristor [36] [37], which is more area-compact and energy-efficient compared with the traditional CMOS one.

3 Adaptive Memory-enhanced Time Delay Reservoir

3.1 Memory Property of Standard Time Delay Reservoir

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 (3) can be rewritten as:

$$h_i^{(k)} = \frac{1}{1+\theta}\mathcal{B}h_i^{(k)} + \frac{\theta}{1+\theta} f(\mathcal{B}^\tau {h_i}^{(k)}, x_i^{(k)}). \tag{5}$$

According to [20], without loss of generality, assume that the linear activation and output functions are identity. Therefore, we can get:

$$h_i^{(k)} = \left[I - \frac{1}{\theta+1}\mathcal{B} - \frac{\theta}{\theta+1}\mathcal{B}^\tau\right]^{-1} \frac{\theta}{\theta+1} x_i^{(k)}. \tag{6}$$

The inverse calculation in Equation 6 could be decomposed as:

$$\sum_{j=0}^{\infty}\left[-\left(I - \frac{1}{\theta+1}\mathcal{B}\right)^{-1}\left(-\frac{\theta}{\theta+1}\mathcal{B}^\tau\right)\right]^j \left(I - \frac{1}{\theta+1}\mathcal{B}\right)^{-1}. \tag{7}$$

The first term in Equation 4 could be transferred into 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:

$$A_j = \left(\frac{1}{\theta+1}\right)^{3j}\left(\frac{\theta}{\theta+1}\right)^{j+1} W_{yh}, \tag{8}$$

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

3.2 Structure of our Proposed Memory-enhanced TDR

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:

$$D\left(h^{(t)};\lambda\right) = \left[((I-\mathcal{B})^{\tau+\lambda} - I)\right] h^{(t)}, \tag{9}$$

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:

$$\dot{h}^{(t)} = -h^{(t)} + f\left(h^{(t-\tau)}, D\left(h^{(t)};\lambda\right), x^{(t)}\right). \tag{10}$$

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:

$$D\left(h^{(t)};\lambda\right)_i = \sum_{j=1}^{m}\left[h_i^{(t-(\mathrm{j}+\lambda_j)\tau)}\right]. \tag{11}$$

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 (10) with integration step θ, the procedure of memory-enhanced TDR could be formulated as:

$$\begin{cases} y^{(k)} = W_{yh} h_i^{(k)} + \varepsilon^{(k)} \\ h_i^{(k)} = \frac{1}{\theta+1} h_{i-1}^{(k)} + \frac{\theta}{\theta+1} f\left(h_i^{(k-1)}, D\left(h^{(k)};\lambda\right)_i, x^{(k)}\right) \\ D\left(h^{(k)};\lambda\right)_i = \sum_{j=1}^{m}\left[h_{\lambda_j\cdot\tau}^{(k-j)}\right], \end{cases} \tag{12}$$

where f is a nonlinear function, and there is the error $\varepsilon^{(k)}$ between the output $\hat{y}^{(k)}$ and the target $y^{(k)}$. This error forms a sequence of independent and identically distributed (i.i.d.) random vectors over time. We also illustrate the procedure of Equation (19) in Fig. 1. As shown in Fig. 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;
- 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.

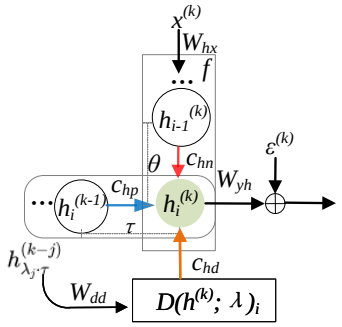


Fig. 1. Reservoir state of memory-enhanced TDR. $W_{hx} \in \mathbb{R}^{p\times q}$ and $W_{yh} \in \mathbb{R}^{p\times q}$ represent the input weights and output weights respectively; c_{hp}, c_{hn}, and c_{hd} represent the transfer factors between self-dependency, neighboring-dependency and long-term dependency to current states, respectively.

Fig. 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 current reservoir

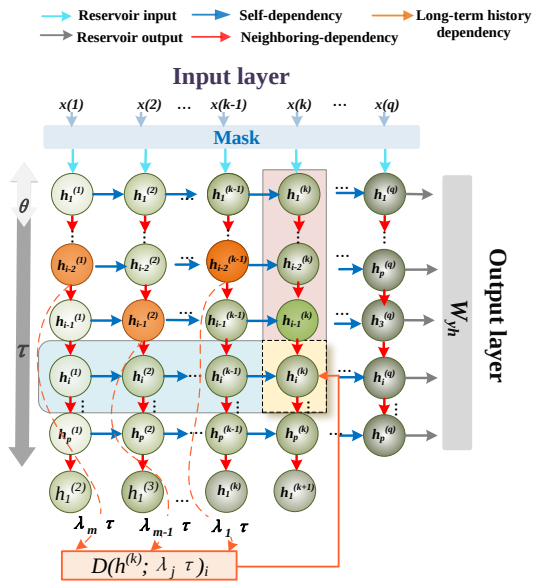


Fig. 2. The discrete form of our proposed memory-enhanced TDR.

state for different tasks. The details of this will be introduced in Section 3.3.

According to Figs. 1 and 2, the reservoir input is the masked input, where the mask is $W_{hx} \in \mathbb{R}^{p\times q}$. 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 with non-zero mean to be zero; such property is convenient for eliminating the intercept in ridge regression. 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:

$$W_{yh} := \arg\min_{W} \left(\sum_{i=1}^{M} ||h_i^{(t)^{\mathsf{T}}} \times W - y_i^{(t)}||^2 + \eta||W||^2 \right) \tag{13}$$

where $||W||^2$ refers to a regularisation term to prevent overfitting, and η controls its intensity. In order to optimize this problem, ridge regression [38] has been applied, whose solution could be given by:

$$W_{yh} = (HH^{\mathsf{T}} + \eta H)^{-1} Hy. \tag{14}$$

3.3 Optimization of Memory-enhanced TDR using PSO

As Section 3.2 shows, the enhanced memory behaves as the weighted accumulation of the previous states in different layers. $\lambda = (\lambda_1, \lambda_j, ..., \lambda_m)'$ indicates the memory-enhanced degree of the long-term memory that existed in different previous layers (from 1st to the m-th layer respectively). However, confronted with tasks with different memory dependencies, there will be $(\lambda_1, \lambda_j, ..., \lambda_m)'$ to be optimized making the memory-enhanced degree adaptive. However, this is an NP-hard problem.

PSO, one of the famous swarm intelligence optimization algorithms, searches for optimal solutions using a population of particles [39]. 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:

$$\begin{aligned} \boldsymbol{v_i}^{(t+1)} = w \cdot \boldsymbol{v_i}^{(t)} + c_1 \cdot rand() \cdot (\boldsymbol{p}_{i_{best}} - \boldsymbol{p_i}^{(t)}) \\ + c_2 \cdot rand() \cdot (g_{best} - \boldsymbol{p_i}^{(t)}), \end{aligned} \tag{15}$$

$$\boldsymbol{p_i}^{(t+1)} = \boldsymbol{p_i} + \boldsymbol{v_i}^{(t+1)}, \tag{16}$$

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 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) as explained in Section 5.2. 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, 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 Eq. (15), and particle $\boldsymbol{p_i}$ will be updated by Eq. (16). 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.

Further verification and analysis of the proposed adaptive memory-enhanced TDR for both of the short-term and long-term dependency tasks will be introduced in Section 5.

4 Memristive Implementation of the Adaptive Memory-enhanced TDR

4.1 Dynamic Memristor Model

In software RC, a reservoir can perform nonlinear transformations of the input signals, and project them to a high-dimensional space. According to Tanaka et al. [4], some of the memristive devices or systems are capable of exhibiting nonlinear dynamic behavior. In this work, we apply a $Ti/TiO_x/TaO_y/Pt$-based dynamic memristor model

Algorithm 1: Pseudo code of PSO optimization of the degree of memory enhancement for TDR

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

```
Initialise population p;
while max_Iter not met do
    for each individual p_i in p do
        Generate memory enhanced TDR with p_i;
        Compute velocity v_i of each individual by Eq. (15);
        Update individual p_i by Eq. (16);
        Calculate fitness_i of individual p_i by f;
    end
    Set: local_best_fitness ⟵ max({fitness_i})
    if local_best_fitness > global_best_fitness then
        p_global_best ⟵ p_local_best
        global_best_fitness ⟵ local_best_fitness
    end
end
Return: p_global_best
```

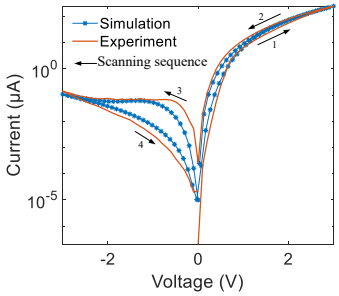


Fig. 3. I–V hysteresis curves of $Ti/TiO_x/TaO_y/Pt$-based dynamic memristor model, where the experimental data comes from [36].

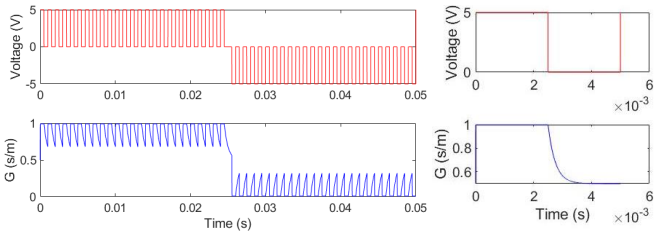


Fig. 4. Conductance change of the dynamic memristor.

proposed by Zhong et al. [36], which is equipped with a fading memory (forgetting effect) to construct the nonlinear dynamic behavior. Its mathematical model is defined as:

$$I = KGV^3, \tag{17}$$

$$G(t) = G_0 + r(G(t-1) - G_0) + \frac{a\,|V|}{a\,|V| + 1}(G_{th} - G(t)). \tag{18}$$

Where I, V, and $G(t)$ represent the output current, input voltage, and the conductance at time step t, respectively. K and G_{th} are the parameters varied with V, which can be depicted as:

$$\begin{cases} K = sign(V)K_p + sign(-V)K_n \\ G_{th} = sign(V), \end{cases} \tag{19}$$

where $sign$ represents a sign function:

$$\begin{cases} sign(x) = 1 & \text{if} \quad x > 0 \\ sign(x) = 0 & \text{if} \quad x \leq 0. \end{cases} \tag{20}$$

Fig. 3 shows the I–V hysteresis curves of the dynamic memristor model. The solid line represents the experiment data sampling by the voltage scan, the dotted line represents the simulation data by simulating the memristor mathematical model, and the arrows indicate the direction of the voltage scan. The output current of the dynamic memristor model will be sampled as the reservoir states.

Fig. 4 shows the conductance change of the dynamic memristor. As we can see, the conductance of the dynamic memristor will decrease nonlinearly with the positive voltage switching from 5V to 0V, and the conductance will increase nonlinearly with the negative voltage switching from -5V to 0V. Considering the nonlinearity and fading memory shown in Figs. 3 and 4, we will employ the dynamic memristor as the reservoir and the current of the memristor will be used as the reservoir states in our work.

The parameters of the memristor models we applied in our work are all listed in Table 1.

4.2 Memristor-based Delay Element (MDE)

One single dynamic memristor can be used to construct nonlinear node exhibiting nonlinear dynamic behavior with fading memory. Instead of a dynamic memristor with the short-term memory, we need memristors with nonvolatile memory to construct the programmable memristor-based delay element. HP memristor model is the one that has this nonvolatile property and has been widely applied to the programmable memristive unit [16]. Therefore, we have also used the HP model to construct the programmable memristive delay element. As Section 2.1 illustrated, the classic TDR only has one type of delay from last state $x(t-1)$, which may limit the predictive performance for the long memory dataset. In order to enhance the memory capacity of MTDR, the corresponding delay element are required to MTDR. By manipulating the current starved inverters and a memristor-based programming unit, the memristor-based delay element (MDE) is shown in Fig. 5. The MDE is composed of two inverters INV_1, INV_2, AND, and a memristive programming unit (MPU).

In this work, we applied our proposed memristive reconfigurable unit [16] as the MPU in the delay element. This MUP is composed of 4 transistors and one memristor, by which the memristance R_m could be tuned by *Configuration_signal* and *Control_signal*. The MDE works in two phases, which are configuration phase and operation phase, respectively. Specifically, when *Control_signal* is positive, the S_1 will connect to operation signal and the delay element will work in the operation phase. When the *Control_signal* is negative, the S_1 will connect to *Configuration_signal* and the delay element will work in configuration phase. And different length of applied *Configuration_signal* will lead to different memristance. According to Elmore delay model [40], the delay from low to high can be formulated as:

$$t_{pLH} = 0.69(R_{eqn1} + R_{eqn2} + R_{eqn3} + R_m)C_L, \tag{21}$$

TABLE 1
The parameters of two memristor models

Dynamic Memristor	Value	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	$10nm$

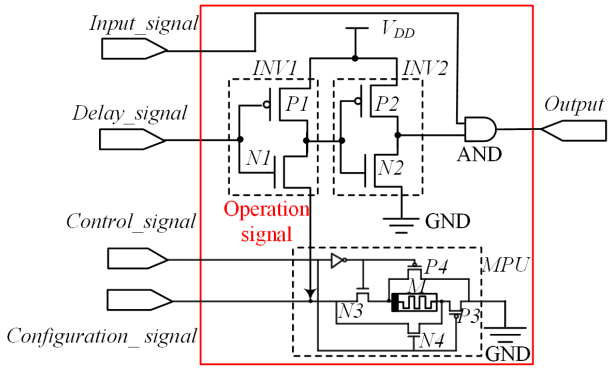


Fig. 5. Schematic diagram of memristor-based delay element.

where R_m is the memristance of M, the equivalent on-resistance R_{eqn1}, R_{eqn2} and R_{eqn3} can be calculated by the following equation [41] :

$$R_{eq} = -0.5V_{DD}\int_{V_{DD}}^{V_{DD}/2} VdV/I_{DSAT}(1+\lambda V) \approx 3V_{DD}/4I_{DSAT}(1-7\lambda/9), \tag{22}$$

where V_{DD} is the supply voltage, I_{DSAT} is the transistor's current in saturation region, and λ is the channel length modulation factor. The memristor M used in the delay element is a HP memristor model with threshold, which is given by:

$$V(t) = (R_{on}\frac{x(t)}{D} + R_{off}(1-\frac{x(t)}{D}))i(t), \tag{23}$$

$$\frac{dx(t)}{dt} = \frac{\mu_v R_{on}}{D}i(t), \tag{24}$$

where R_{on} and R_{off} represent the minimum and maximum memristance, respectively. D, μ_v and $x(t)$ denote the effective length of memristor, dopant mobility rate and the length of memristor's doped region, respectively. The parameters of memristor M are listed in Table 1. The value μ_v is varied under different applied voltages, which can be depicted as follows:

$$\begin{cases} \mu_v = \mu_0 & \text{if} \quad |V(t)| \geq V_{th} \\ \mu_v = 0 & \text{if} \quad |V(t)| < V_{th}. \end{cases} \tag{25}$$

Once the applied voltage $V(t)$ exceeds the threshold voltage V_{th}, the dopant mobility rate will be μ_0, otherwise, it will be 0 and the memristance will remain constant. With this threshold property, we set that the amplitude of *Configuration_signal* exceeds the threshold, so that the memristance could be configured during the configuration phase, while the memristance will be unchanged during the operation phase.

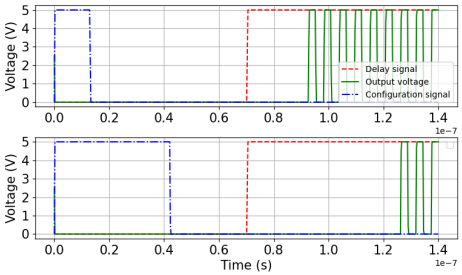


Fig. 6. Simulation result of the memristor-based delay element.

According to Equation (24), the relationship between resistance and charge ($q(t)$) can be written as:

$$R_m(t) = \left(R_{on}^2\frac{\mu_v}{D^2} - \frac{R_{off}\mu_v R_{on}}{D^2}\right)q(t) \tag{26}$$

As $R_{on} \ll R_{off}$, Equation (26) could be simplified as:

$$R_m(t) = \left(-\frac{R_{off}\mu_v R_{on}}{D^2}\right)q(t). \tag{27}$$

Therefore, there will be a differential equation of $R_m(t)$ and current $i_m(t)$:

$$\frac{dR_m(t)}{dt} = \left(-\frac{R_{off}\mu_v R_{on}}{D^2}\right)i_m(t). \tag{28}$$

The amplitude of *Configuration_signal* is denoted as V_{conf} that exceeds the threshold of memristor V_{th}, so that its duration Δt_{conf} can lead to different memristance. We assume that the initial memristance of M is $R_m(0)$. Then, according to Ohm law, Equation (28) is solved as:

$$R_m(t) = \sqrt{(R_m(0))^2 \pm \left(-\frac{2R_{off}\mu_v R_{on}}{D^2}\right)V_{conf}\Delta t_{conf}}. \tag{29}$$

Therefore, the memristance of M will be varied with different duration of *Configuration_signal*, t.

Fig. 6 shows the simulation results of the memristor-based delay element, where two sub-figures show two scenarios with different duration of *Configuration_signal*. The simulation contains both of the configuration and operation phases, where the first half period (0-70ns) is the configuration phase, and the last half period (70-140ns) represents the operation phase. The red dash line represents the delay signal, the blue dashdotted line represents the *Configuration_signal*, and the green solid line represents the output voltage of the delay element. According to Equation (29) and Equation (21), the larger duration of *Configuration_signal* will lead to larger memristance and incur the larger delay further.

4.3 Architecture of Adaptive Memristive Memory-enhanced TDR

The architecture diagram of the memristive implementation of memory-enhanced TDR is shown in Fig. 7. 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

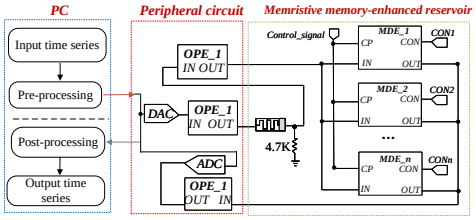


Fig. 7. The memristive implementation framework of proposed adaptive memory-enhanced TDR.

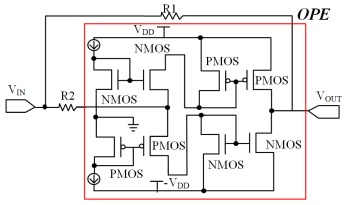


Fig. 8. Circuit schematic of OPE module. PMOS uses M2SJ136 model and NMOS uses M2SK1029 model in JPWRMOS library.

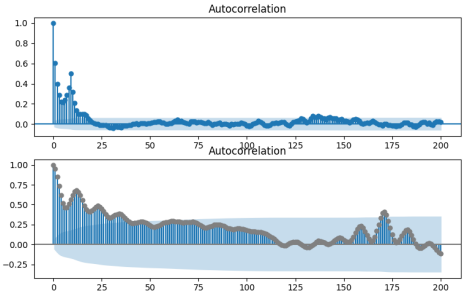


Fig. 9. Autocorrelation plot of Narma10 (top) and Nonlinear audio dataset (bottom).

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 Fig. 8 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 Fig. 10. Executing the memristive memory-enhanced TDR consists of 8 steps, related steps also have been marked in the Fig. 10:

- **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 120μs 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).
- **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*).
- **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 Experiments

5.1 Datasets

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 [20][25]. We visualize the autocorrelation plots of one short-term memory dataset (Narma10) and one long-term memory dataset (Nonlinear audio) in Fig. 9.

5.1.1 *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 [25]. This dataset is from [42], 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 [1], 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, 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 with obvious long memory effect:

$$(1-0.7B+0.4B^2)(1-B)^{0.4}Y_t = (1-0.2B)\varepsilon_t \quad (30)$$

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

1. https://pkg.yangzhuoranyang.com/tsdl/

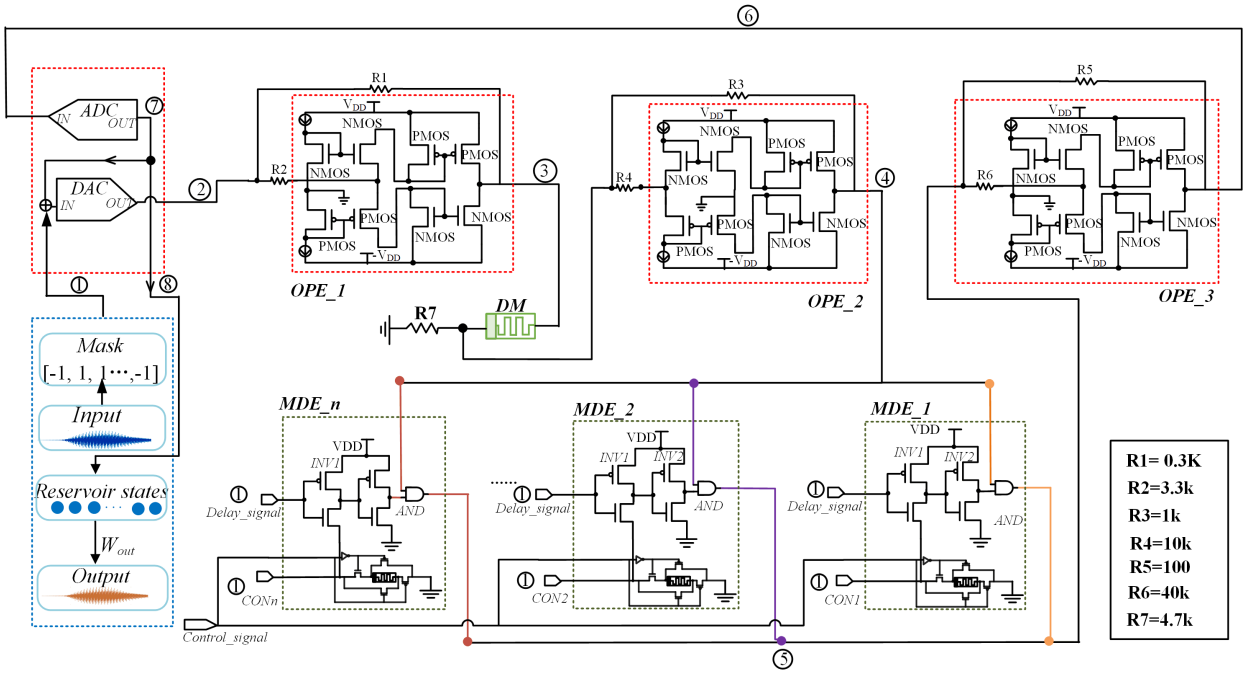


Fig. 10. Memristive implementation of adaptive memory-enhanced TDR.

5.1.2 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 [2], which is a cross-cut through periodic to chaotic intensity pulsations of a real laser. 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:

$$y(t+1) = 0.3y(t) + 0.05y(t)\sum_{i=0}^{9} y(t-i) + 1.5s(t-9)s(t) + 0.1, \tag{31}$$

$$y(t+1) = \tanh\left(0.3y(t) + 0.05y(t)\sum_{i=0}^{19} y(t-i) + 1.5s(t-19)s(t) + 0.1\right). \tag{32}$$

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:

$$x(n+1) = y(n) - 1.4x(n)^2, \tag{33}$$

$$y(n+1) = 0.3x(n) + w(n). \tag{34}$$

2. http://web.cecs.pdx.edu/~mcnames/DataSets/index.html

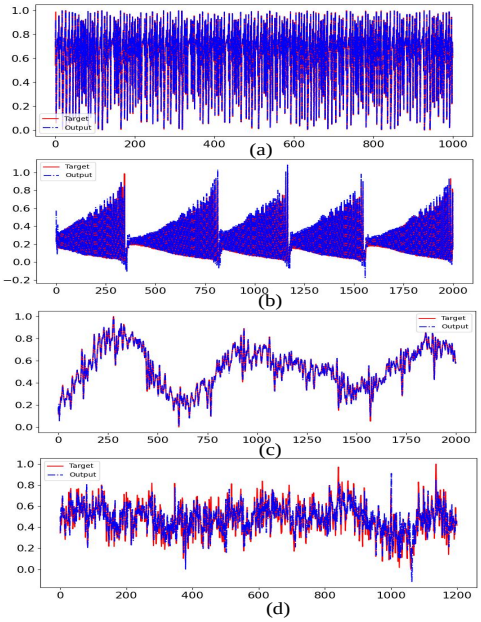


Fig. 11. Actual predicted outputs of our proposed adaptive memory-enhanced TDR vs corresponding targets. (a) Hénon Map; (b) Santa set-A; (c) Nonlinear audio; (d) ARFIMA series.

5.2 Comparisons 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 [43], Deep ESN [23], standard TDR

[10], vanilla LSTM [44], vanilla RNN [45], a variant of LSMT (mLSTM) and a variant of RNN (mRNN) [20].

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

$$RMSE = \sqrt{\left\langle \|\hat{y}(t) - y(t)\|^2 \right\rangle}, \tag{35}$$

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 [20] 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 [23]. 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.

Fig. 11 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 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.

TABLE 2
Parameter Settings

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

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 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 [43] and standard TDR [10], but also over other heavier approaches such as RNNs [45], LSTMs [44], deep-ESN [23], and variants of RNN and LSTM [20].

5.3 Comparisons of different hardware reservoirs

We also compare our proposed memristive TDR with other hardware reservoirs, where the comparisons contain predictive performance comparison (shown in Table 3), and hardware performance comparison (shown in Table 5).

From Table 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 summarizes the design specification of our proposed memristor-based memory-enhanced TDR and other state-of-the-art reservoir computing designs, where works [8] and [5] are related to the hardware implementations of ESN, work [46] is the hardware implementation of LSM, works [12] [11] [37] [36] focus on the hardware implementations of TDR. We compare the number of components used

TABLE 3
Prediction performance comparison of different models

Datasets	Nar.10	Nar.20	SantaA	SantaD	Hénon Map	Average (AvRan.*)	Non. audio	Tree	DJI	Arfima	Average (AvRan.)	Average (AvRan.)
Type	Short-memory						Long-memory					S&L
Software model												
RNN [45]	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 [44]	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 [20]	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 [20]	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 [43]	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 [23]	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 [10]	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 [36]	0.0980	0.0907	0.0741	0.0463	0.0192	-	0.0117	0.2762	0.2391	1.150	-	-
Bai's [11]	0.0683	-	-	-	-	-	-	-	-	-	-	-
Ours	**0.0372**	**0.0463**	**0.0170**	**0.0237**	**0.0046**	-	**0.0097**	**0.1212**	**0.1096**	**0.0860**	-	-

* 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.

TABLE 4
The results of Mann–Whitney U tests.

Method	RNN	LSTM	mRNN	mLSTM
P-value	0.0170	0.0136	0.0318	0.0211
Method	ESN	Deep-ESN	TDR	
P-value	0.0318	0.0067	0.0318	

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 Fig. 12. Taking FPGA-based ESN [8] 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 [11], 57 Mosfets and 3 capacitors are required to construct a reservoir, which is more compact than the ESN reservoir hardware [8][5]. 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 [11] 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 [37] [36] 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.

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[37]. The other two memristor-based TDRs [37] [36] 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 [36]. However, the easiest reservoir architecture [36] also limits the memory ability for predicting long-term dependency tasks (see the hardware reservoir comparisons in Table 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 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. Therefore, when opting for one of these approaches, there is a trade-off between prediction performance and power consumption.

6 Conclusion

We proposed an adaptive memory-enhanced TDR for time series prediction and its memristive implementation. In terms of the prediction performance and circuit performance, our proposed method and its memristive implementation outperformed prior models and hardware reservoirs across the long (short) memory datasets. By making use of an adaptive higher-order delay unit, the memory of TDR can be enhanced and the enhanced degree for each layer can

TABLE 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 [8]	FPGA	0	37n*	n	-	No	-
Mosfet crossbar-based ESN [5]	PCB	0	16n	0	-	No	-
CMOS-based LSM [46]	IC	0	24n	3n	-	No	-
BBD-based TDR [12]	PCB	0	128	1	-	No	-
CMOS-based TDR [11]	IC	0	57	3	$526\mu w$	No	-
Memristor-based TDR [37]	IC	1	0	0	$300\mu w$	No	-
Memristor-based TDR with mask [36]	IC	1	0	0	$50\mu w$	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 -.

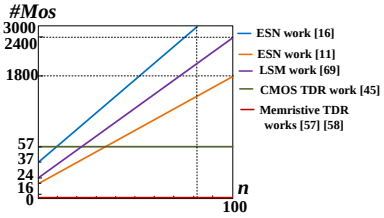


Fig. 12. The relationship between the number of neuron and the number of Mosfets.

be optimized adaptively according to different given tasks using PSO. As shown experimentally, with this design our proposed adaptive memory-enhanced TDR obtained better predictive performance on both short-term and long-term memory datasets.

Making full use of the potential of hardware-friendly feature of TDR, the memristive implementation of proposed adaptive memory-enhanced TDR was also presented in this paper. In this implementation, a dynamic memirstor is used as a nonlinear node, and the memristor-based delay element is used to construct the adaptive higher-order delay unit, which expands the memory capacity of the hardware reservoir unlike other prior works. By virtue of nano-size and low energy efficiency of memristor, the proposed memristive implementation also shows its superiority on the circuit area and power consumption compared with traditional device-based reservoirs. Future work will focus on improving the model's predictive performance and the hardware implementation of the memristive output layer to achieve a fully analog memristive TDR with adaptive enhanced memory.

REFERENCES

[1] C. Merkel, "Design of a time delay reservoir using stochastic logic: A feasibility study," in *Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN)*, Anchorage, AK, USA, 2017, pp. 2186–2192.

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

[3] F. Denis-Le Coarer, M. Sciamanna, A. Katumba, M. Freiberger, J. Dambre, P. Bienstman, and D. Rontani, "All-optical reservoir computing on a photonic chip using silicon-based ring resonators," *IEEE J. Select. Topics Quantum Electron.*, vol. 24, no. 6, pp. 1–8, 2018.

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

[5] Y. Kume, S. Bian, and T. Sato, "A tuning-free hardware reservoir based on mosfet crossbar array for practical echo state network implementation," in *Proc. IEEE 25th Asia and South Pac. Des. Autom. Conf. (ASP-DAC)*, Beijing, China, 2020, pp. 458–463.

[6] S. Wolf, D. Awschalom, R. Buhrman, J. Daughton, v. S. von Molnár, M. Roukes, A. Y. Chtchelkanova, and D. Treger, "Spintronics: a spin-based electronics vision for the future," *Science*, vol. 294, no. 5546, pp. 1488–1495, 2001.

[7] W. Maass and H. Markram, "On the computational power of circuits of spiking neurons," *J. Comput. Syst. Sci.*, vol. 69, no. 4, pp. 593–616, 2004.

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

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

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

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

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

[13] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, 2008.

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

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

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

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

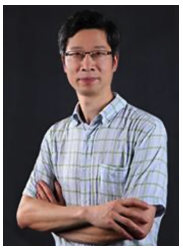
[18] M. D. Skowronski and J. G. Harris, "Automatic speech recognition using a predictive echo state network classifier," *Neural Netw.*, vol. 20, no. 3, pp. 414–423, 2007.

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

[20] J. Zhao, F. Huang, J. Lv, Y. Duan, Z. Qin, G. Li, and G. Tian, "Do rnn and lstm have long memory?" in *Proc. 37th Int. Conf. Mach. Learn.*, Virtual Event, 2020, pp. 11 365–11 375.
[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[22] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
[23] Q. Ma, L. Shen, and G. W. Cottrell, "Deepr-esn: A deep projection-encoding echo-state network," *Inf. Sci.*, vol. 511, pp. 152–171, 2020.
[24] L. Grigoryeva, J. Henriques, L. Larger, and J.-P. Ortega, "Stochastic nonlinear time series forecasting using time-delay reservoir computers: Performance and universality," *Neural Netw.*, vol. 55, pp. 59–71, 2014.
[25] A. Greaves-Tunnell and Z. Harchaoui, "A statistical investigation of long memory in language and music," in *Proc. 36th Int. Conf.Mach. Learn.*, Long Beach, California, USA, 2019, pp. 2394–2403.
[26] J. Beran, Y. Feng, S. Ghosh, and R. Kulik, *Long-Memory Processes*. Springer, 2016.
[27] F. Sabzikar, A. I. McLeod, and M. M. Meerschaert, "Parameter estimation for artfima time series," *J. Stat. Plan. Inference*, vol. 200, pp. 129–145, 2019.
[28] T. Lin, B. G. Horne, P. Tino, and C. L. Giles, "Learning long-term dependencies in narx recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 7, no. 6, pp. 1329–1338, 1996.
[29] O. Levy, K. Lee, N. FitzGerald, and L. Zettlemoyer, "Long short-term memory as a dynamically computed element-wise weighted sum," *arXiv preprint arXiv:1805.03716*, 2018.
[30] R. DiPietro, C. Rupprecht, N. Navab, and G. D. Hager, "Analyzing and exploiting narx recurrent neural networks for long-term dependencies," *arXiv preprint arXiv:1702.07805*, 2017.
[31] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio, "Architectural complexity measures of recurrent neural networks," *Adv. in Neur. Inf. Process. Syst.*, vol. 29, pp. 1822–1830, 2016.
[32] M. L. Alomar, V. Canals, N. Perez-Mora, V. Martínez-Moll, and J. L. Rosselló, "Fpga-based stochastic echo state networks for time-series forecasting," *Comput. Intell. Neurosci.*, vol. 2016, 2016.
[33] Q. Wang, Y. Li, and P. Li, "Liquid state machine based pattern recognition on fpga with firing-activity dependent power gating and approximate computing," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Montréal, QC, Canada, 2016, pp. 361–364.
[34] B. Schrauwen, D. Verstraeten, and J. Van Campenhout, "An overview of reservoir computing: theory, applications and implementations," in *Proc. 15th Eur. Symp. on Artif. Neur. Netw.*, Bruges, Belgium, 2007, pp. 471–482.
[35] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2635–2649, 2015.
[36] Y. Zhong, J. Tang, X. Li, B. Gao, H. Qian, and H. Wu, "Dynamic memristor-based reservoir computing for high-efficiency temporal signal processing," *Nature Commu.*, vol. 12, no. 1, pp. 1–9, 2021.
[37] J. Moon, W. Ma, J. H. Shin, F. Cai, C. Du, S. H. Lee, and W. D. Lu, "Temporal data classification and forecasting using a memristor-based reservoir computing system," *Nature Electronics*, vol. 2, no. 10, pp. 480–487, 2019.
[38] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.
[39] Y. Del Valle, G. K. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, and R. G. Harley, "Particle swarm optimization: basic concepts, variants and applications in power systems," *IEEE Trans. Evol. Comput.*, vol. 12, no. 2, pp. 171–195, 2008.
[40] A. B. Kahng and S. Muddu, "An analytical delay model for rlc interconnects," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 12, pp. 1507–1514, 1997.
[41] N. H. Weste and D. Harris, *CMOS VLSI design: a circuits and systems perspective*. Pearson Education India, 2015.
[42] G. Holzmann, "Reservoir computing: a powerful black-box framework for nonlinear audio processing," in *Proc. 12th Int. Conf. Digit. Audio Eff. (DAFx)*, Como, Italy, 2009.
[43] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.
[44] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," 2014.
[45] C. L. Giles, G. M. Kuhn, and R. J. Williams, "Dynamic recurrent neural networks: Theory and applications," *IEEE Transs Neural Netws*, vol. 5, no. 2, pp. 153–156, 1994.
[46] S. Roy, A. Banerjee, and A. Basu, "Liquid state machine with dendritically enhanced readout for low-power, neuromorphic vlsi implementations," *IEEE Trans. Biomed. Circuits Syst.*, vol. 8, no. 5, pp. 681–695, 2014.

Xinming Shi received the B.S. degree in electronic engineering from Wuhan University of Technology, Wuhan, China, in 2016. She received the M.S. degree in School of Artificial Intelligence & Automation from Huazhong University of Science and Technology, Wuhan, China, in 2019. She is currently pursuing her Ph.D. degree in Computer Science at the University of Birmingham, U.K., in collaboration with Southern University of Science and Technology, Shenzhen, China.

Leandro L. Minku Dr. Leandro L. Minku is an Associate Professor (Senior Lecturer) at the School of Computer Science, University of Birmingham (UK). Prior to that, he was a Lecturer in Computer Science at the University of Leicester (UK). He received the PhD degree in Computer Science from the University of Birmingham (UK) in 2010. Dr. Minku's main research interests are machine learning in non-stationary environments / data stream mining, online class imbalance learning, ensembles of learning machines and computational intelligence for software engineering. Among other roles, Dr. Minku is Associate Editor-in-Chief for Neurocomputing, Senior Editor for IEEE Transactions on Neural Networks and Learning Systems, Associate Editor for Journal of Systems and Software, Associate Editor for Empirical Software Engineering journal. He was also the General Chair for the International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2019 and 2020), and Co-chair for the Artifacts Evaluation Track of the International Conference on Software Engineering (ICSE 2020).



Xin Yao obtained his Ph.D. in 1990 from the University of Science and Technology of China (USTC), MSc in 1985 from North China Institute of Computing Technologies and BSc in 1982 from USTC. He is a Chair Professor of Computer Science at the Southern University of Science and Technology, Shenzhen, China, and a part-time Professor of Computer Science at the University of Birmingham, UK. He is an IEEE Fellow and was a Distinguished Lecturer of the IEEE Computational Intelligence Society (CIS). He was the President (2014-15) of IEEE CIS and the Editor-in-Chief (2003-08) of IEEE Transactions on Evolutionary Computation. His major research interests include evolutionary computation, ensemble learning, and their applications to software engineering. His work won the 2001 IEEE Donald G. Fink Prize Paper Award; 2010, 2016 and 2017 IEEE Transactions on Evolutionary Computation Outstanding Paper Awards; 2011 IEEE Transactions on Neural Networks Outstanding Paper Award; and many other best paper awards at conferences. He received a prestigious Royal Society Wolfson Research Merit Award in 2012, the IEEE CIS Evolutionary Computation Pioneer Award in 2013 and the 2020 IEEE Frank Rosenblatt Award.