A Brain-Inspired Hardware Architecture for Evolutionary Algorithms Based on Memristive Arrays

ZILU WANG, XINMING SHI, and XIN YAO, Southern University of Science and Technology, China

Brain-inspired computing takes inspiration from the brain to create energy-efficient hardware systems for information processing, capable of performing highly sophisticated tasks. Systems built with emerging electronics, such as memristive devices, can achieve gains in speed and energy by mimicking the distributed topology of the brain. In this work, a brain-inspired hardware architecture for evolutionary algorithms is proposed based on memristive arrays, which can realize sparse and approximate computing as a result of the parallel analog computing characteristic of the memristive arrays. On this basis, an efficient evolvable brain-inspired hardware system is implemented. We experimentally show that the approach can offer at least a four orders of magnitude speed improvement. We also use experimentally grounded simulations to explore fault tolerance and different parameter settings in the implemented hardware system. The experimental results show that the evolvable hardware system, implemented based on the proposed hardware architecture, can continuously evolve toward a better system even if there are failures or parameter changes in the memristive arrays, demonstrating that the proposed hardware architecture has good adaptability and fault tolerance.

CCS Concepts: • **Hardware** → **Emerging architectures**; **Emerging simulation**; *Simulation and emulation*; System-level fault tolerance; • **Computer systems organization** → Neural networks;

Additional Key Words and Phrases: Memristor, evolutionary algorithms, brain-inspired architecture, parallel analog computing, circuit implementation

ACM Reference format:
Zilu Wang, Xinming Shi, and Xin Yao. 2023. A Brain-Inspired Hardware Architecture for Evolutionary Algorithms Based on Memristive Arrays. *ACM Trans. Des. Autom. Electron. Syst.* 28, 5, Article 82 (September 2023), 32 pages.
https://doi.org/10.1145/3598421

This work was supported by the Young Scientists Fund of the National Natural Science Foundation of China (grant 62206121), the Postdoctoral Science Foundation of China (grant 2021M701578), the Research Institute of Trustworthy Autonomous Systems (RITAS), the Guangdong Provincial Key Laboratory (grant 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (grant 2017ZT07X386), and the Shenzhen Science and Technology Program (grant KQTD2016112514355531).
Authors' address: Z. Wang, X. Shi, and X. Yao, Southern University of Science and Technology, 1088 Xueyuan Avenue, Shenzhen, China, 518055; emails: wangzl@sustech.edu.cn, xxs972@cs.bham.ac.uk, xiny@sustech.edu.cn.

1084-4309/2023/09-ART82
https://doi.org/10.1145/3598421

1 INTRODUCTION

Brain-inspired computing, which emulates the structure and working principle of the biological brain, has emerged as a new computing paradigm to improve computing efficiency [1, 2]. In contrast, traditional computing systems are generally built on CMOS (complementary metal oxide semiconductor) transistors where internal communication may be limited by storage capacity and transmission efficiency, leading to high energy consumption and computational burden [3–6]. It is therefore necessary to explore beyond-CMOS devices to build higher-performance brain-inspired computing systems [7]. As an emerging two-terminal memory device, the memristor [8–10] has many excellent properties [11–14], such as intrinsic dynamics, analog behavior, nonvolatility, high speed, low power, high density, and great scalability. Memristors are therefore particularly suitable for serving as neurons or synapses to construct brain-inspired hardware systems [15–17]. In addition, memristors can be easily integrated into crossbar array architectures, which naturally support parallel in-memory computing and high-density storage [18, 19]. For example, the memristive crossbar array has the ability to carry out multiply-add operations in parallel in the analog domain, which can be utilized to accelerate some computing-intensive modules in an artificial neural network with high throughput at low energy and area consumption [5, 20, 21]. It can be seen that researching memristor-based brain-inspired hardware system is both feasible and necessary.

In addition to powerful computing abilities, another salient characteristic of the human brain is its adaptability, which allows the brain to engage in flexible and robust information processing [22]. Implementing an efficient and biomimetic memristor-based brain-inspired hardware system requires high-density, parallel storage, and computation capabilities. Incorporating design features for good self-adaptability and fault tolerance within dynamic environments would improve and accelerate the development process of such a system [2, 23]. Evolvable hardware is designed to adapt to changes in task requirements or environments by dynamically and autonomously reconfiguring the hardware structure [24, 25]. This capacity for adaptation is achieved by evolutionary algorithms [26, 27], which are a set of widely used and highly effective bio-inspired algorithms that loosely simulate the laws of natural genetic evolution. Evolutionary algorithms can also help the system adjust parameters according to the environment changes with adaptive learning and high fault tolerance [28–30]. For example, Samajdar et al. [31] proposed a hardware-software prototype of an evolutionary algorithm-based learning system (named *GENESYS*) that enables continuous learning through neural network evolution in hardware, demonstrating good energy efficiency and computational speed advantages, which can contribute to the large-scale deployment of intelligent devices with autonomous learning capabilities at the edge. Inspired by the preceding advantages of evolvable hardware and the features of evolutionary algorithms, it is highly advantageous to combine the conceptual ideas underlying these evolutionary systems and incorporate their intrinsic qualities into the design of memristor-based brain-inspired hardware systems [32, 33] that have both efficient computing and biomimetic adaptive learning abilities.

To effectively integrate the ideas of evolutionary algorithms into the designs of brain-inspired hardware systems, the implementation of the computing processes in evolutionary algorithms is designed to be achieved at the hardware level. Hardware implementations of a range of evolutionary algorithms, such as a hardware-based genetic algorithm through very large scale integration architecture in **Field-Programmable Gate Arrays (FPGAs)** [34–36], have been proposed to improve processing speed. In addition, hardware-oriented evolutionary algorithms are proposed [37, 38] to adapt to the hardware implementation, reducing hardware cost and increasing application flexibility. However, most research is based on digital circuits with FPGAs, which are usually constructed with complex circuit architectures, facing the issues of scaling limit and

energy consumption [3, 39]. In comparison, memristors can store multi-bit information with continuously adaptable conductance, unlike the binary states "0" and "1" in traditional digital circuits. The memristor has advantages such as non-volatility, fast programming, low programming energy, and a compact footprint, which makes it possible to efficiently realize the hardware implementations of evolutionary algorithms through parallel analog circuits [8–14]. In this way, memristive devices will help build efficient and biomimetic brain-inspired hardware systems. However, there are few studies on memristor-based hardware implementation for evolutionary algorithms in the current literature, and the possible factors for such results are due to the lack of memristive devices with corresponding performance, appropriate circuit mapping mechanisms, or effective computing methods [1, 5, 40, 41]. Therefore, in this work, we present the following:

(1) A brain-inspired hardware architecture for evolutionary algorithms based on memristive arrays is proposed. It is realized by a bottom-up circuit design approach—that is, from the cell modules of the bottom level to the functional modules of the middle level to the overall architecture of the top level.
(2) An efficient evolvable brain-inspired hardware system is implemented based on the proposed hardware architecture. The simulation results show that due to effective mappings between memristive circuits and evolutionary algorithms, the computing speed of the system is improved by analog computing that is intrinsically sparse and approximate in a relatively low operating frequency.
(3) The good adaptability of the proposed hardware architecture is also confirmed by performing the experiments with fault tolerance and different parameter settings experiments on the evolvable brain-inspired hardware system. It is observed that the system can always evolve adaptively toward a better direction based on the current input information—that is, it can realize online learning on-the-fly based on spiking signals.

The rest of this article is organized as follows. Section 2 introduces the memristor model used in this work. Section 3 proposes a brain-inspired hardware architecture for evolutionary algorithms based on memristive arrays. Section 4 proposes a memristive circuit implementation method based on the proposed hardware architecture in Section 3, so as to realize an efficient evolvable brain-inspired hardware system. Section 5 details the simulations of the implemented hardware system, the performance analyses, and comparisons with others. Section 6 presents the conclusions drawn from this work and discusses further works.

2 DRIFT SPEED ADAPTIVE MEMRISTOR MODEL

As an important member of the emerging non-volatile memory, memristor is a simple metal-insulator-metal (*MIM*) sandwich structure that can achieve resistance switching (from a high resistance state (*HRS*) to a low resistance state (*LRS*)) under external voltage biases. In other words, a memristor is a two-terminal ionic device that uses resistance states to represent information. It has rich switching dynamics that make it a suitable element to mimic the functions of neuron and synapse in human brain, thus allowing the simulation of a brain-inspired hardware system.

In this work, a non-volatile memristor named the **Drift Speed Adaptive Memristor (DSAM)** model is used [42], which matches the conductive property of the AgInSbTe memristor [43]. Its i-v relationship is

$$v(t) = (R_{\text{off}} - x \cdot \triangle R) \cdot i(t), \tag{1}$$

where the state variable x is a normalized width of the conducting layer, whose derivative is matched to memristor current-voltage data, and its range is [0, 1]. Additionally, $\triangle \text{R} = (R_{\text{off}}\text{-}R_{\text{on}})$, with R_{off} and R_{on} indicating the maximum resistance and minimum resistance of the memristor, respectively (i.e., the bounds of device resistance), with their corresponding state variables being

Table 1. Parameters of the DSAM Model

Parameters	a_{on}	a_{off}	p_{on}	p_{off}	k_{on}	k_{off}	$V_{on}(V)$	$V_{off}(V)$	$R_{on}(\Omega)$	$R_{off}(\Omega)$
Setting	125	175000	2.0	−0.01	100	125000	0.6	−0.6	3450	162220

$x = 0$ and $x = 1$. Therefore, the derivative of the state variable x can be expressed as follows [42]:

$$\frac{dx}{dt} = \begin{cases} k_{on} \cdot \triangle R \cdot i(t) \cdot f(x), & v(t) > V_{on}, \\ 0, & V_{off} \le v(t) \le V_{on}, \\ k_{off} \cdot \triangle R \cdot i(t) \cdot f(x), & v(t) < V_{off}, \end{cases} \tag{2}$$

$$f(x) = \begin{cases} (a_{on} \cdot (1-x))^{p_{on}}, & v(t) > 0, \\ (a_{off} \cdot x)^{p_{off}}, & v(t) \le 0, \end{cases} \tag{3}$$

where $f(x)$ is a speed adaptive state variable function. V_{on} and V_{off} represent positive and negative threshold voltages, respectively, and only when the applied input voltage meets the threshold condition will the state of the memristor be changed. a_{on}, a_{off}, p_{on}, and p_{off} are scaling parameters responsible for determining the drift speed of indirect effects. k_{on} and k_{off} represent the average ion mobility of oxygen vacancies, similarly to the parameter "μ_v" in the HP model [44].

Based on the roles of the preceding parameters in the DSAM model and the functions to be realized by using it in the proposed brain-inspired hardware architecture for evolutionary algorithms, the parameter setting for the DSAM model in this work is shown in Table 1. The memristance change curve of the DSAM model under this kind of parameter setting is shown in Figure 1. When a forward voltage that satisfies the threshold voltage V_{on} is applied to the memristor, the resistance of the memristor starts to drop and stabilizes when it reaches R_{on}. After that, as shown in Figure 1, a reverse voltage that satisfies the threshold voltage V_{off} is applied at 5 us, and the resistance of the memristor begins to rise to R_{off} to stabilize. Therefore, the resistance of the memristor can be written to an appropriate value by controlling the duration of the applied input voltage across the memristor. Naturally, different parameter settings can alter the memristance switching speed, memristance range, and so forth, thus affecting the performance of the realized circuit system, which will be analyzed in later sections.

3 BRAIN-INSPIRED HARDWARE ARCHITECTURE FOR EVOLUTIONARY ALGORITHMS

According to the computing principles of evolutionary algorithms such as evolutionary programming, a framework for the brain-inspired hardware architecture is proposed, as shown in Figure 2, by referring to the structures of biological neural networks. On this basis, a brain-inspired hardware architecture based on memristive arrays for evolutionary algorithms is designed, as shown in Figure 3. This design uses a bottom-up circuit design method, from the cell module in the bottom layer to the functional module in the middle layer and then to the overall architecture in the top layer. In this way, the memristive arrays can be used as core computing units to realize brain-inspired analog computing in parallel, enabling the hardware architecture to run in a sparse and approximate analog computing manner. The details are described as follows.

3.1 Brief Introduction to Evolutionary Algorithms

Biological evolution refers to the evolution processes of occurrence and development of all life forms. As described in Darwin's theory of evolution, species in nature will evolve based on the way of "natural selection and survival of the fittest", so as to propagate and adapt traits

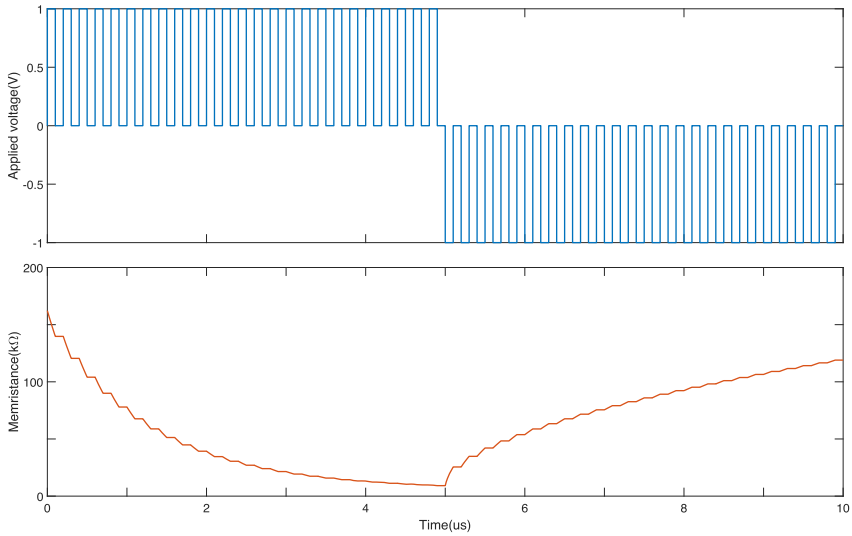

Fig. 1. Memristance change curve of the DSAM model based on the parameter setting shown in Table 1. It can be seen that when a forward voltage that satisfies the threshold voltage V_{on} is applied to the memristor, the resistance of the memristor starts to drop and stabilizes when it reaches R_{on}. After that, a reverse voltage that satisfies the threshold voltage V_{off} is applied at 5 us, and the resistance of the memristor begins to rise to R_{off} to stabilize.

associated with current optimal individuals while they respond to environmental changes throughout their evolutionary lifetimes [45]. Inspired by the theory of biological evolution, researchers have proposed a series of evolutionary algorithms, which are a class of heuristic stochastic optimization algorithms and can be used to simulate natural evolution processes by considering mutation, recombination, and natural selection [46]. In other words, evolutionary algorithms are stochastic search methods that operate on a population of solutions by simulating the evolution of species observed in nature at a high level of abstraction. Although there are many kinds of evolutionary algorithms, such as the genetic algorithm, evolutionary programming, and so on [27], they can be abstracted into the following four steps:

(1) Generate a set containing several initial solutions, and become a population.
(2) Some new solutions are generated based on the current population by means of mutation or crossover.
(3) A new population is formed by removing some relatively poor solutions from the current population and the new solutions generated.
(4) Return to the second step and repeat until a set of conditions (e.g., the population converging to some solutions) for termination is satisfied.

In general, just like the evolutionary programming algorithm [47, 48] as shown in Table 2, the algorithm inspired by biological evolution can be summarized into two major steps: one is to mutate/crossover the solutions in the current population, and the other one is to select the next generation from the mutated/crossed (i.e., offspring) and the current solutions. In other words, the mutation, recombination (i.e., crossover), and the natural selection are the basic operations of evolutionary algorithms. Inspired by evolvable hardware [24, 25], the system based on evolutionary algorithms can adapt to different problems in different environments and can obtain satisfactory and effective solutions in most cases, and for which such a system is characterized by properties of

Table 2. Pseudo-Code of Evolutionary Programming (EP)

```
begin EP
  //start with an initial time
  t := 0;
  //initialize a usually random population of individuals
  Initialize population P(t);
  //evaluate fitness of all initial individuals of population
  Evaluate P(t);
  //Test for termination criterion (time, fitness, etc.)
  while not done do
    //perturb the whole population stochastically to generate the offspring individuals
    P'(t) := mutate [P(t)];
    //evaluate fitness of all offspring individuals of population
    Evaluate P'(t);
    //stochastically select the survivors from actual fitness
    P(t+1) := select [P(t), P'(t)];
    //increase the time counter
    t := t + 1
  od
end EP
```

self-organization, self-adaptation, and so on. Building on previous work on biological neural networks [1, 4], a brain-inspired hardware architecture based on memristive arrays that inherently and functionally embodies the processes of evolutionary algorithms is proposed in the next section.

3.2 Brain-Inspired Hardware Architecture for Evolutionary Algorithms Based on Memristive Arrays

According to the preceding computing principles of evolutionary programming and the structures of biological neural networks, a brain-inspired hardware architecture for evolutionary algorithms is designed, as shown in Figures 2 and 3. The evolutionary module in Figure 2 (the part marked with a red dashed box) focuses on simulating the core computing operations of evolutionary algorithms by mainly operating the memristive array, allowing the proposed brain-inspired hardware architecture run in a sparse and approximate analog computing manner. Concretely, the individuals of the population in the evolutionary module are associated with synaptic weights akin to the synapses in the biological neural network. Additionally, the synaptic weights are represented by the memristances of memristors in the memristive array, which facilitate sparse and approximate computing due to the parallel analog computing characteristic of the memristive array. As well, the fitness are calculated based on the learning/optimization goals in the hardware architecture, which vary according to different functions in output neurons of the biological neural network. The iterative update mechanism of the evolutionary module is established by referring to the processing mechanism based on different spiking signals in input neurons.

Let $\mathbb{N}$ be the set of natural numbers $\{1, 2, 3, \ldots\}$. Let i, j, m, and n be natural numbers that we use as indices in the following presentation, satisfying $m>1$, $n>1$, $1\leq i<m$, and $1\leq j<n$. Let $m\times n$ be the population size. The evolutionary module in Figure 2 can be described as follows:

$$Parent_{ij} = \boldsymbol{f}(Weight_{ij}, Update_{ij}), \tag{4}$$

$$Offspring_{ij} = \boldsymbol{q}(Parent_{ij}, Mutation_{ij}), \tag{5}$$

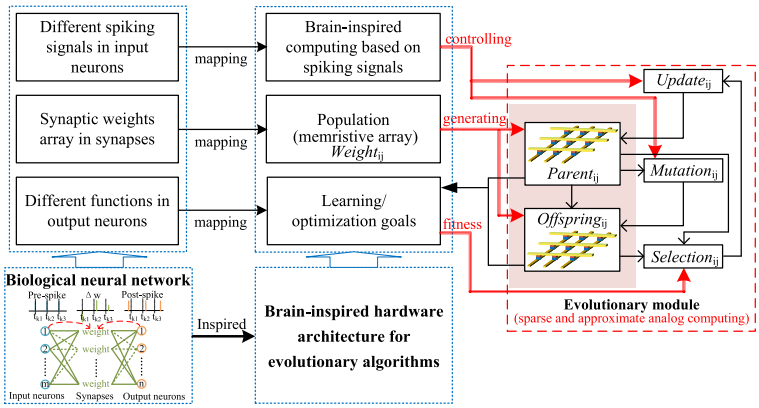

Fig. 2. The framework of the brain-inspired hardware architecture for evolutionary algorithms. The part marked with a red dashed box (evolutionary module) is used to simulate the computing operations of evolutionary algorithms in a sparse and approximate analog computing way, which is mainly realized through the support of memristive arrays.

$$Mutation_{ij} = \boldsymbol{g}(Parent_{ij}), \tag{6}$$

$$Selection_{ij} = \boldsymbol{s}(\boldsymbol{k}(Parent_{ij}), \boldsymbol{k}(Offspring_{ij})), \tag{7}$$

$$Update_{ij} = \boldsymbol{l}(Selection_{ij}), \tag{8}$$

where $Weight_{ij}$ represents the synaptic weight of one node in the brain-inspired hardware architecture—that is, the memristor of the i^{th} row and the j^{th} column in the memristive array. $Parent_{ij}$ represents the j^{th} gene in the i^{th} parent. $Offspring_{ij}$ represents the j^{th} gene in the i^{th} offspring. $Mutation_{ij}$ represents the mutagenic factor of the i^{th} row and the j^{th} column. $Selection_{ij}$ represents the select result of the i^{th} row and the j^{th} column. $Update_{ij}$ represents the updating factor of the i^{th} row and the j^{th} column, which is used to update the corresponding memristor in the memristive array. $\boldsymbol{f}(\cdot)$ represents the function that produces parents, $\boldsymbol{q}(\cdot)$ represents the function that produces offspring, $\boldsymbol{g}(\cdot)$ represents the mutation function, $\boldsymbol{k}(\cdot)$ represents the fitness function, $\boldsymbol{s}(\cdot)$ represents the selection function, and $\boldsymbol{l}(\cdot)$ represents the function that produces corresponding updating factor.

In this way, formulas (4) and (5) can be used to respectively describe the parents and offspring information in each generation. The update of each generation of parents ($Parent_{ij}$) in formula (4) is related to the synaptic weights ($Weight_{ij}$) in the brain-inspired hardware architecture and the updating factors ($Update_{ij}$) that are produced from the corresponding select results. The update of each generation of offspring ($Offspring_{ij}$) in formula (5) is related to the parents ($Parent_{ij}$) and the mutagenic factors ($Mutation_{ij}$). The mutagenic factor ($Mutation_{ij}$) is produced from the mutation function $\boldsymbol{g}(\cdot)$. The select result ($Selection_{ij}$) is produced from the selection function $\boldsymbol{s}(\cdot)$, which is related to the results of fitness function $\boldsymbol{k}(\cdot)$ based on parents and offspring. In the following, combined with the proposed hardware architecture for evolutionary algorithms, an evolvable memristive brain-inspired hardware system is implemented.

In general, the design of the brain-inspired hardware architecture based on memristive arrays for evolutionary algorithms employs a bottom-up circuit design method, as shown in Figure 3. In detail, the first is the design of cell modules in the bottom layer, which involves creating regular circuits based on memristors to meet the performance requirements of brain-inspired computing.

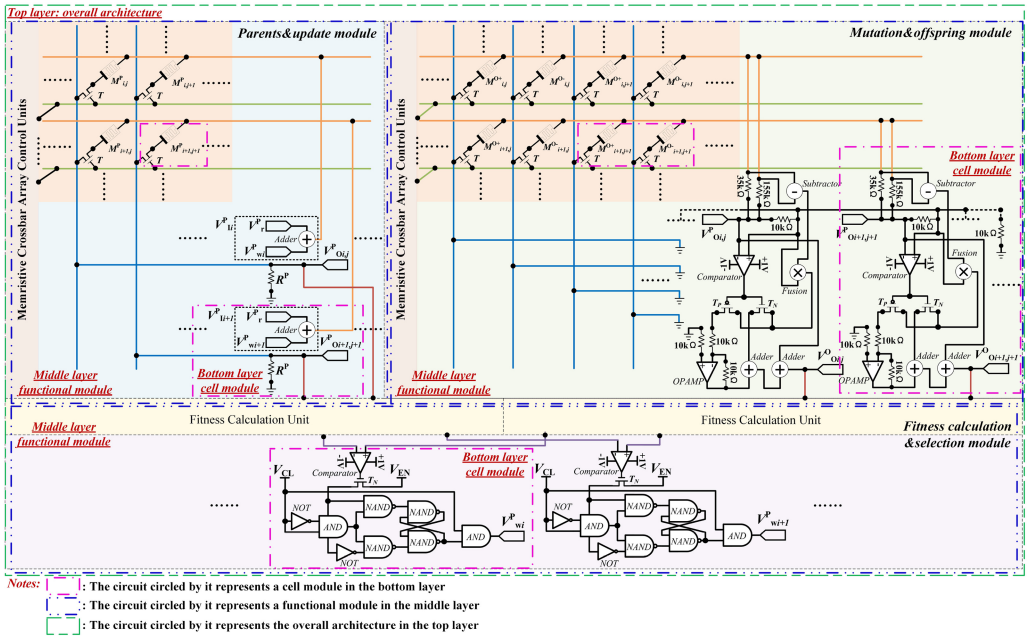

Fig. 3. The schematic of the brain-inspired hardware architecture for evolutionary algorithms based on memristive arrays, which is designed by using a bottom-up circuit design method. It mainly includes the *parents&update* module, the *mutation&offspring* module, and the *fitness calculation&selection* module. The *parents&update* module is mainly used to produce $m \times n$ parents $V^{\mathrm{P}}_{\mathrm{O}ij}$, the *mutation&offspring* module is mainly used to produce $m \times n$ offspring $V^{\mathrm{O}}_{\mathrm{O}ij}$, and the *fitness calculation&selection* module is mainly used to produce m update/write signals $V^{\mathrm{P}}_{\mathrm{w}i}$. The memristive array is mainly used in the *parents&update* module to represent the population for evolution—that is, the processes of evolution are carried out by operating the corresponding memristors in the memristive array based on the information from the other two modules.

The second is the design of functional modules in the middle layer, where core functional modules, such as mutation and selection from the evolutionary module of Figure 2, are realized by mapping the regular memristive circuits with the principles of biological evolution. The third is the design of the overall architecture in the top layer, achieved by cascading the designed functional modules based on the computing principles of the evolutionary algorithms and the structures of biological neural networks. The detailed circuit implementation methods for these modules will be elaborated in Section 4. To better control each memristor in the array, avoid leakage current, and address other issues, the memristive arrays in Figure 3 adopt the form of a 1T1R crossbar [49, 50], which has also been widely studied in practical memristor materials. In this way, the memristive array can be used as core computing units to realize brain-inspired parallel analog computing, enabling the brain-inspired hardware architecture for evolutionary algorithms designed based on it to run in a sparse and approximate analog computing way. This approach is expected to improve the computing speed.

4 EVOLVABLE BRAIN-INSPIRED HARDWARE SYSTEM IMPLEMENTATION

On the basis of the proposed brain-inspired hardware architecture for evolutionary algorithms shown in Figure 3, an evolvable brain-inspired hardware system is implemented, which can be used to efficiently solve the corresponding optimization/prediction problems through the way of

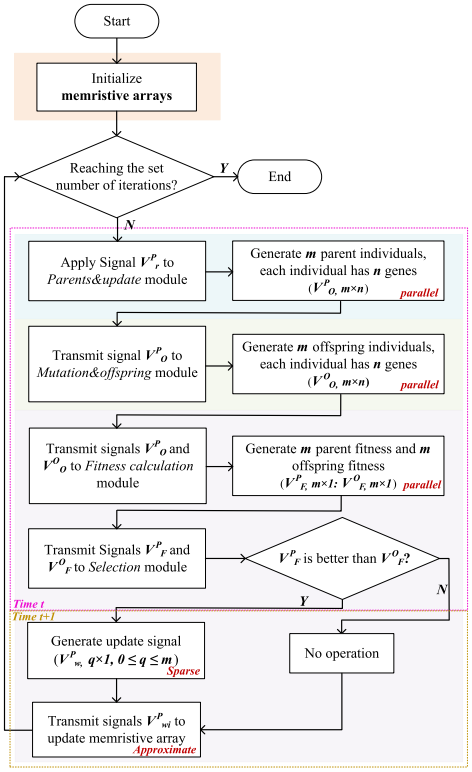

Fig. 4. The work flow chart of the evolvable brain-inspired hardware system. The light blue area corresponds to the operations in the *parents&update* module of Figure 3. The light green area corresponds to the operations in the *mutation&offspring* module of Figure 3. The light purple area corresponds to the operations in the *fitness calculation&selection* module of Figure 3. The orange area corresponds to the memristive arrays in Figure 3. The area enclosed by pink dotted line refers to the operations to be performed at time t, and all operations are performed in parallel. The area enclosed by the yellow dotted line area refers to the operations to be performed at time $t + 1$, and all operations are performed by using a brain-inspired way (sparse and approximate) in parallel.

brain-inspired computing. It mainly includes the *parents&update* module, the *mutation&offspring* module, and the *fitness calculation&selection* module. Their circuit implementations are shown later in Figures 5 through 7, which are the schematics of their cell modules in bottom layers (functional modules in middle layers are the superposition and combination of multiple cell modules, whereas the overall architecture in the top layer, i.e., the hardware system, connects multiple functional modules according to the execution principle). The execution principle of this evolvable brain-inspired hardware system is shown in Figure 4, which indicates the mapping mechanisms from the evolutionary algorithms to the memristive circuit implementations.

First, in the *parents&update* module, the initial population comes from the synaptic weights in the brain-inspired memristive hardware system—that is, the resistances of memristors in the memristive array. These synaptic weights need to be updated or optimized based on the goal of the system. The parents are produced from the population, which will be updated (adopting the way of approximate computing) based on the results of the *fitness calculation&selection* module in parallel. Specifically, the synaptic weights in there are mapped in an $m \times n$ memristive array.

In other words, the parents population in each generation has m individuals, and each parent individual has n genes.

Second, in the *mutation&offspring* module, the offspring will be produced through mutation operation based on the current parents from the *parents&update* module in parallel, and Cauchy mutation is adopted in there [48]. Similarly, the offspring population in each generation has m individuals, and each offspring individual has n genes as well.

Third, in the *fitness calculation&selection* module, the fitness of each individual in parents and offspring will be calculated, which is used to select the current optimal individuals. There are m fitness of parents and m fitness of offspring, and they will be compared one by one in parallel, so as to determine which genes in the parents need to be updated (i.e., the updating process is sparse). In other words, if the fitness of an individual in the offspring is better than that of the parents, an update/write signal will be produced to adjust the corresponding synaptic weight—that is, the corresponding memristance in the memristive array will be changed. Otherwise, there will be no update/write signal, so the memristance in the memristive array will not need to be changed.

In general, as for the population (i.e., the $m \times n$ memristive array), only the non-optimal genes in parents (i.e., the corresponding memristors in the memristive array) need to be updated to make them change toward a required better direction consistently. In this way, the computing processes are sparse, so as to effectively improve the computing efficiency. Additionally, as for the operations of updating the non-optimal genes (i.e., to change the memristances of the corresponding memristors in the memristive array), it is not necessary to accurately change the corresponding genes (i.e., the corresponding memristances in the memristive array) to the current optimal values, but to change in a better direction, so that the updating operations in there is approximate, which can also improve the computing efficiency. Besides, in the preceding execution principles, the individuals and their corresponding genes in all parents and offspring can be computed and updated in parallel, so as to speed up the evolutionary computing process. Due to the preceding reasons, the evolvable brain-inspired hardware system implemented based on the proposed hardware architecture can realize efficient brain-inspired computing operations. The details of the circuit implementations are described as follows.

4.1 *Parents&update* Module Implementation

The *parents&update* module is mainly used to produce and update the parents population based on the synaptic weights from the memristive array and the updating signals from the *fitness calculation&selection* module. As shown in Figure 5, the parents population is represented by an $m \times n$ memristive array M_{ij}^{P}, and the output voltage of the parents $V_{\mathrm{O}ij}^{\mathrm{P}}$ are

$$V_{\mathrm{O}ij}^{\mathrm{P}} = V_{Ii}^{P} \cdot \frac{R_{\mathrm{P}}}{M_{ij}^{\mathrm{P}} + R_{\mathrm{P}}}, \tag{9}$$

$$V_{Ii}^{P} = V_{\mathrm{r}}^{\mathrm{P}} + V_{\mathrm{w}i}^{\mathrm{P}}, \tag{10}$$

where i indicates the i^{th} individual in m parent individuals and j indicates the j^{th} gene in n genes of the one individual. M_{ij}^{P} represents the memristance of the memristor in the i^{th} row and the j^{th} column of the memristive array, the parameter settings of memristors in there are shown in Table 1, and the initial memristances of them need random initialization. The read signal V_{r} is used to produce the corresponding parent gene signal $V_{\mathrm{O}ij}^{\mathrm{P}}$ based on the corresponding memristance of memristor in the memristive array. The write signal $V_{\mathrm{w}i}^{\mathrm{P}}$ comes from the *fitness calculation&selection* module and is used to update the corresponding memristance of memristor in the memristive array—that is, to carry out the write operation for the corresponding memristor. The models of the discrete components (memristors and transistors) in the *parents&update* module are

Table 3. Models of the Discrete Components in the Evolvable Brain-Inspired Hardware System

Component	*Position*	*Model*
M^P	Memristors in *parents&update* module (generating parents)	*Non-volatile memristor (e.g., DSAM model [42])*
M^O	Memristors in *mutation&offspring* module (generating offspring)	*Non-volatile memristor (e.g., DSAM model)*
M_F	Memristors in *mutation&offspring* module (realizing *Fusion* function)	*Non-volatile memristor (e.g., DSAM model)*
T_N	Transistors in the evolvable brain-inspired hardware system	*N-channel enhancement type (e.g., 2N7000)*
T_P	Transistors in the evolvable brain-inspired hardware system	*P-channel enhancement type (e.g., 2N6806)*

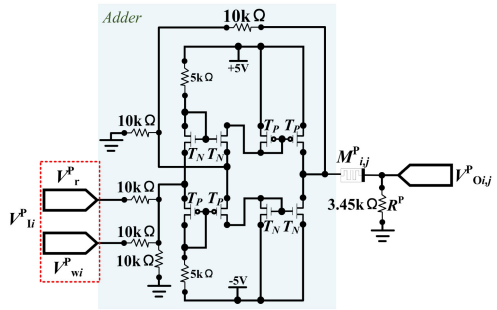

Fig. 5. The circuit implementation schematic of one cell module in the bottom layer of the *parents&update* module. The whole *parents&update* module (i.e., one functional module in the middle layer) is composed of $m \times n$ cell modules. The parents population is generated by the $m \times n$ memristive array M^{P}_{ij}, and $V^{\mathrm{P}}_{\mathrm{O}ij}$ represents the j^{th} gene in the i^{th} parent individual. This module is mainly used to produce and update the parents population based on the synaptic weights from the memristive array and the updating signals from the *fitness calculation&selection* module.

shown in Table 3, and the parameter settings of resistors are marked in Figure 5. As for these resistance parameters, they are determined based on the desired functions to be implemented and are utilized as the corresponding peripheral circuits to assist the computation of related memristive circuit modules. Therefore, starting from the intended functions (i.e., the addition operation as shown in formula (10)), the resistance parameters in Figure 5 are also comprehensively considered in terms of the resistance range of the memristive devices, compatibility with other circuit components, and power consumption, to ultimately determine the required functionality, performance, and energy efficiency for the cell module in the bottom layer of the *parents&update* module. The resistance parameters in Figures 6 and 7 are also determined in this manner.

4.2 *Mutation&offspring* Module Implementation

The *mutation&offspring* module is mainly used to produce the offspring by mutating the parents based on Cauchy mutation, which is inspired by fast evolutionary programming [48]. As shown in Figure 6, $M^{\mathrm{O}+}_{ij}$ and $M^{\mathrm{O}-}_{ij}$ are the core computing components for fitting Cauchy distribution, and the fitting curve is shown in Figure 8. The following *Subtractor*, *Fusion*, *Comparator*, *Adder*, and so forth are used to process the signals from these two memristors, so as to complete the Cauchy mutation operations on the parents to produce offspring. The *Subtractor* is used to perform the subtraction calculation for the two input signals. The *Fusion* is used to complete the signal fusion operation for the two input signals by using the memristor M_{F}, which is similar to the analog multiplier. The parameter setting of M_{F} memristor in *Fusion* is shown in Table 1, and the initial memristance of it is set to *HRS* R_{off}. The *Comparator* is used to realize the comparison operation for the two input signals. In detail, if the input signal of port "+" is greater than the input signal

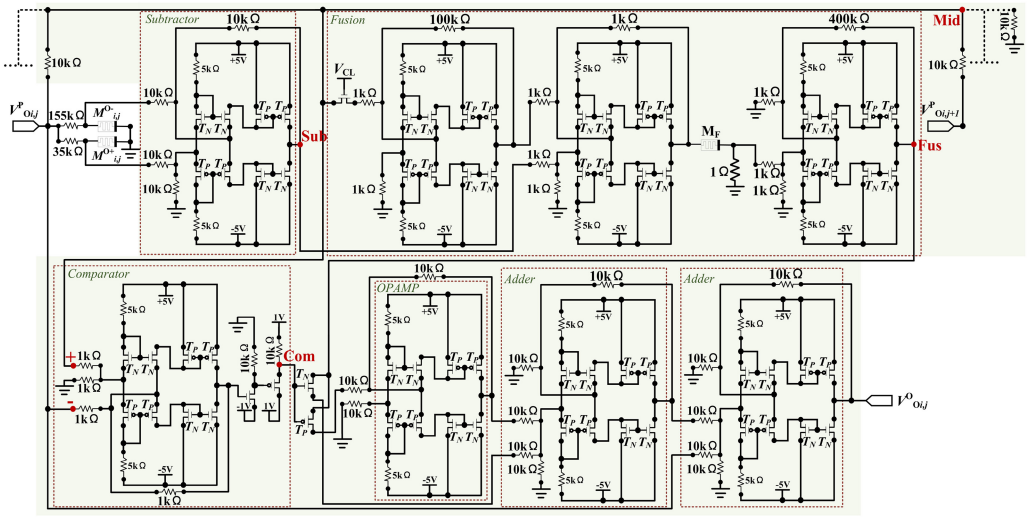

Fig. 6. The circuit implementation schematic of one cell module in the bottom layer of the *mutation&offspring* module. The whole *mutation&offspring* module (i.e., one functional module in the middle layer) is composed of $m \times n$ cell modules. This functional module is mainly used to produce offspring by mutating the parents based on Cauchy mutation, which is mainly realized by using M_{ij}^{O+} and M_{ij}^{O-}. V_{Oij}^{O} represents the j^{th} gene in the i^{th} offspring individual, which is generated by mutating the corresponding signal of parent gene V_{Oij}^{P}.

of port "−," *Comparator* will output a voltage of 1V, and otherwise it will output a voltage of −1V. The *OPAMP* is used to simulate the function of operational amplifier, so as to complete the reverse proportion calculation operation for the input signal in combination with the connected resistances. The *Adder* is used to realize the addition calculation for the two input signals.

With the cooperation of the preceding components, the output voltages of offspring V_{Oij}^{O} are

$$V_{Oij}^{O} = V_{Oij}^{P} + [V_{O(\text{Fus}_{ij})} \cdot \boldsymbol{sgn}(V_{O(\text{Com}_{ij})})], \tag{11}$$

$$V_{O(\text{Fus}_{ij})} = \frac{400 \cdot (V_{O(\text{Mid}_{i})} + 100 \cdot V_{O(\text{Sub}_{ij})})}{M_{\text{F}}}, \tag{12}$$

$$V_{O(\text{Sub}_{ij})} = V_{Oij}^{P} \cdot \left[\frac{M_{ij}^{O+}}{(35k\Omega + M_{ij}^{O+})} - \frac{M_{ij}^{O-}}{(155k\Omega + M_{ij}^{O-})} \right], \tag{13}$$

$$V_{O(\text{Mid}_{i})} = \frac{1}{n} \cdot \sum_{j=1}^{n} V_{Oij}^{P}, \tag{14}$$

where i indicates the i^{th} individual in m offspring and j indicates the j^{th} gene in n genes. M_{ij}^{O+} and M_{ij}^{O-} are both memristors that used to fit the Cauchy distribution, and their parameter settings are shown in Table 1; the initial memristances of them are set to *HRS* (R_{off}). The $V_{O(\text{Subij})}$ is the output signal of *Subtractor*, the $V_{O(\text{Fusij})}$ is the output signal of *Fusion*, and the $V_{O(\text{Comij})}$ is the output signal of *Comparator*. The $V_{O(\text{Midi})}$ is the voltage at point "Mid" in Figure 6, which represents the average value of all parent genes in each row. The models of the discrete components (memristors and transistors) in the *mutation&offspring* module are shown in Table 3, and the parameter settings of resistors are marked in Figure 6.

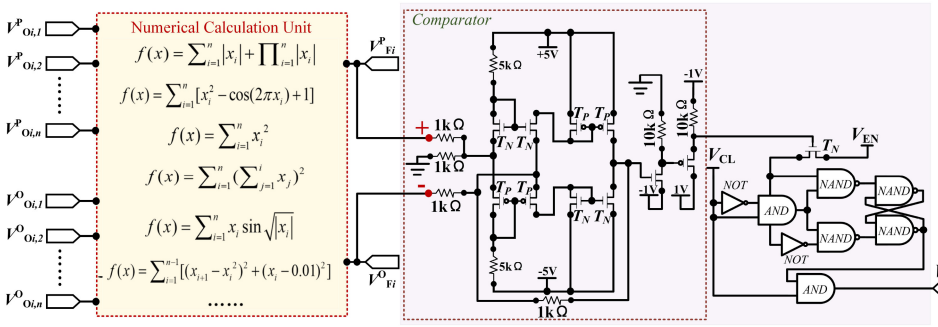

Fig. 7. The circuit implementation schematic of one cell module in the bottom layer of the *fitness calculation&selection* module. The whole *fitness calculation&selection* module (i.e., one functional module in the middle layer) is composed of m cell modules. This functional module is mainly used to calculate and compare the fitness of each individual in parents and offspring, so as to determine whether the update signal is generated. $V_{\mathrm{F}i}^{\mathrm{P}}$ represents the fitness of the i^{th} parents individual. $V_{\mathrm{F}i}^{\mathrm{O}}$ represents the fitness of the i^{th} offspring individual. $V_{\mathrm{w}i}^{\mathrm{P}}$ represents the write signal, which is input to the *parents&update* module and used to update the corresponding parent gene.

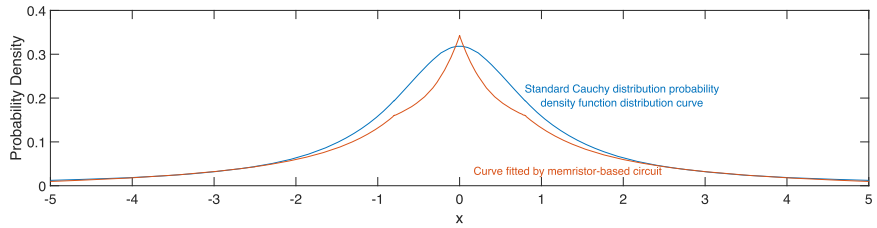

Fig. 8. The fitting curve of the Cauchy distribution by using memristors of $M_{ij}^{\mathrm{O}+}$ and $M_{ij}^{\mathrm{O}-}$ in Figure 6, which is used to carry out the corresponding mutation operation to the parent gene.

4.3 *Fitness calculation&selection* Module Implementation

As shown in Figure 7, the *fitness calculation&selection* module is mainly used to calculate and compare the fitness for each individual in parents and offspring, so as to determine whether the update signal is generated. In this way, the memristive array in the *parents&update* module can be rewritten based on the write signals through the way of brain-inspired computing. As an example, benchmark problems of some unimodal and multimodal optimization functions are considered fitness functions:

$$V_{\mathrm{w}i}^{\mathrm{P}} = \begin{cases} 0\mathrm{V}, \mathrm{V}_{\mathrm{F}i}^{\mathrm{P}} \leq \mathrm{V}_{\mathrm{F}i}^{\mathrm{O}}, \\ 1\mathrm{V}, \mathrm{V}_{\mathrm{F}i}^{\mathrm{P}} > \mathrm{V}_{\mathrm{F}i}^{\mathrm{O}}, \end{cases} \tag{15}$$

$$V_{\mathrm{F}i}^{\mathrm{P}} = \boldsymbol{h}(V_{\mathrm{O}ij}^{\mathrm{P}}), i \in [1, m], j \in [1, n], \tag{16}$$

$$V_{\mathrm{F}i}^{\mathrm{O}} = \boldsymbol{h}(V_{\mathrm{O}ij}^{\mathrm{O}}), i \in [1, m], j \in [1, n], \tag{17}$$

where i indicates the i^{th} individual in m parents or offspring, and j indicates the j^{th} gene in n genes of the one individual in parents or offspring. The $\boldsymbol{h}$ represents the corresponding fitness functions, which can be referred to Table 4. The models of the discrete components (transistors)

i \ j	1	2	3	4	5	6	7	8	9	10
1	63562	132305	88047	59135	152536	142524	90798	102280	96655	36433
2	51279	78218	40044	137501	34372	39319	30553	39596	72626	52844
3	150055	71754	32793	147118	159005	73129	21092	44423	68342	97902
4	45081	99164	116370	38656	22092	50553	54062	70795	84083	17027
5	45124	130627	8089	150925	119405	81026	95303	41123	76301	156360
6	90266	86191	40220	81072	102532	111277	66246	61788	160312	9441
7	143989	148453	129860	19122	45027	56694	111371	25130	117960	20400
8	107247	81910	127140	116977	146934	144902	56505	114390	34856	8299
9	121587	82839	79647	147093	100279	101517	139904	131338	95016	32492
10	41544	144202	8002	81232	30111	158836	116605	82910	78245	12915

Fig. 9. A group of random initial memristances of the memristive array in the *parents&update* module, which are used to produce the first generation of parents. The units of them are ohm (Ω). For example, as for the initial parents population, the first gene in the first individual is represented by the memristor in the first column and the first row M_{11}^{P}, whose memristance is 63562Ω.

in the *fitness calculation&selection* module are shown in Table 3, and the parameter settings of resistors are marked in Figure 7.

5 SIMULATION RESULTS AND ANALYSIS

A series of simulation experiments are carried out to validate the conceptual ideas of the proposed brain-inspired hardware architecture for evolutionary algorithms and demonstrate the efficiency of the evolvable brain-inspired hardware system. Based on the experimental results for the benchmark problems and the prediction problem (as shown later in Figures 10 through 14), it can be seen that the proposed brain-inspired hardware architecture for evolutionary algorithms can realize sparse and approximate computing due to the parallel analog computing characteristic of the memristive arrays. Thus, compared with the traditional software system based on evolutionary algorithms, the evolvable brain-inspired hardware system implemented based on the proposed hardware architecture provides a speed improvement of four to seven orders of magnitude, and the larger the scale of the problem to be solved, the more obvious the speed advantage (as shown in Tables 5 and 6). Additionally, based on the simulation results of the fault tolerance experiments and the experiments with different parameter settings on the evolvable brain-inspired hardware system (as shown later in Figures 15 through 20), it can be seen that the proposed brain-inspired hardware architecture for evolutionary algorithms can evolve adaptively toward a better direction based on the current input information, instead of uncontrollably evolving in a less-than-optimal direction due to an increase in faulty rate or different parameter settings, which indicates that the proposed hardware architecture has high stability and fault tolerance. Meanwhile, it is also found that the evolvable systems implemented based on our hardware architecture with different functions can realize online training through the way of learning on-the-fly based on spiking signals. The detailed descriptions are as follows.

5.1 Simulation Results of Benchmark/Prediction Problems and Fault Tolerance Experiments

As shown in Table 4, these six benchmark problems [48] are simulated with the evolvable brain-inspired hardware system. The definition domains for these benchmark problems are determined by the high and low memristance states of the memristive array. If a wider range of definition domain is required, by using double memristors in each cell of the memristive array, the domain of

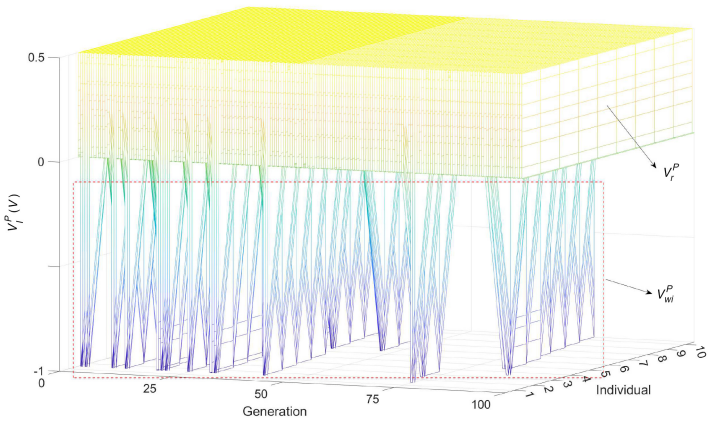

Fig. 10. The simulation results of the input signals $V_{\mathrm{I}i}^{\mathrm{P}}$ in the *parents&update* module, which consist of write signal $V_{\mathrm{w}i}^{\mathrm{P}}$ and read signal $V_{\mathrm{r}}^{\mathrm{P}}$. The write signal $V_{\mathrm{w}i}^{\mathrm{P}}$ is generated based on the *fitness calculation&selection* module, in which can be seen that the write signal infrequently occurs so as to help implement sparse and approximate computing in the *parents&update* and *mutation&offspring* modules. The whole simulation lasted 100 generations, and the whole evolution execution time took 2.0e-5s. So the duration of each generation is 2.0e-7s, the first half (1.0e-7s) in each generation is used to perform the read operation, and the second half (1.0e-7s) is used to perform the write operation.

Table 4. Experimental Settings for Different Benchmark Problems

No.	*Benchmark Problems*	*Population Size (i*n)*	*No. of Generations*
①	$f_{min}(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	10 ×10	100
②	$f_{min}(x) = \sum_{i=1}^{n} [(x_i)^2 - cos(2\pi x_i) + 1]$	10 ×10	100
③	$f_{min}(x) = \sum_{i=1}^{n} (x_i)^2$	10 × 10	100
④	$f_{min}(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$	10 ×10	100
⑤	$f_{min}(x) = \sum_{i=1}^{n} (x_i sin\sqrt{\lvert x_i \rvert})$	10 ×10	100
⑥	$f_{min}(x) = \sum_{i=1}^{n-1} [(x_{i+1} - x_i^2)^2 + (x_i - 0.01)^2]$	10 ×10	100

definition can be extended from the current positive real numbers to the range of positive-negative real numbers. The dimensions n of the functions in Table 4 are set as 10—that is, each individual has 10 genes. The dimensions m of the individuals are set as 10—that is, each generation has 10 individuals. The number of iterations of the proposed system is set as 100 (i.e., evolving 100 generations), and the working frequency is set as 5 MHz (the basis for setting this working frequency will be explained in the next section). In this way, 100 generations will complete the evolution in 2e-5s. The initial parents population—that is, the first generation of parents—come from the initial memristances M_{ij}^{P} of the memristive array, and the one group of random initial memristances is shown in Figure 9 (e.g., M_{11}^{P} = 63562 Ω). The simulation results of these benchmark problems are shown in Figure 10 and later in Figures 11 through 13. The other benchmark problems can also be simulated by adjusting the "Numerical Calculation Unit" in the *fitness calculation&selection* module based on their fitness functions.

In the simulation experiments, the working frequency is set to 5 MHz—that is, the frequency of input signal V_{Ii}^{P} signals (as shown in Figure 5) is 5 MHz. In this way, the duration of the whole evolutionary process is 2.0e-5s, which carries out 100 generations, so the duration of each

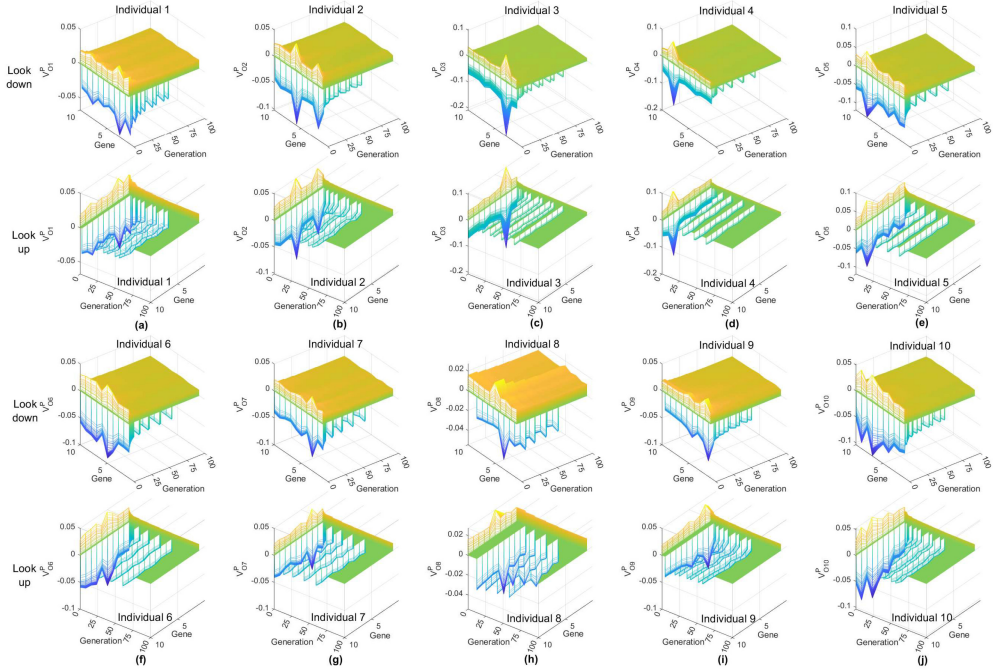

Fig. 11. The simulation results of the parents V_{Oij}^{P} in the *parents&update* module over 100 generations for benchmark problem ①. Subfigures (a) through (j) represent the evolution processes in 100 generations from the first parent individual to the 10th parent individual in turn. Each subfigure shows the evolution processes of 10 genes in corresponding individuals over 100 generations. As for each gene, the duration of the whole evolutionary process is 2.0e-5s and the duration of each generation is 2.0e-7s. Each generation has two stages of reading and writing. The first stage is the read operation (duration is 1.0e-7s), as shown in the subfigure where the vertical axis is positive, which is used to produce the corresponding genes in parent individuals. The second stage is the write operation (duration is 1.0e-7s), as shown in the subfigure where the vertical axis is negative, which is used to complete operations similar to selection by changing the memristance of corresponding memristor in the memristive array. In the writing stage, it can be seen that approximate computing operations are performed only at non-zero V_{Oij}^{P} values, which are sparse.

generation in the whole evolutionary process is 2.0e-7s. As shown in Figure 10, there are two stages in each generation. The first stage is the read operation in the first half of 2.0e-7s (the duration is 1e-7s), which is used to produce the corresponding parents and offspring, and calculate the fitness of the individuals in the parents and the offspring, respectively, as well as compare the fitness of the individual in the parents with the fitness of the individual in the offspring. The second stage is the write operation in the second half of 2.0e-7s (the duration is also 1e-7s), which is used to generate the write signals based on the comparison results from the first stage so as to change the corresponding memristances of memristors in the memristive array, thereby selecting the superior individuals as parents for the next generation.

As shown in Figure 11 and later in Figures 12 and 13, the preceding operations in the first and the second stage are carried out in parallel by taking advantage of the memristive arrays and the proposed bottom-up brain-inspired hardware architecture. In addition, in the second stage, the writing operations are only carried out if the fitness of the individual in the offspring is better than the fitness of the individual in the parents, and the writing operations do not necessarily

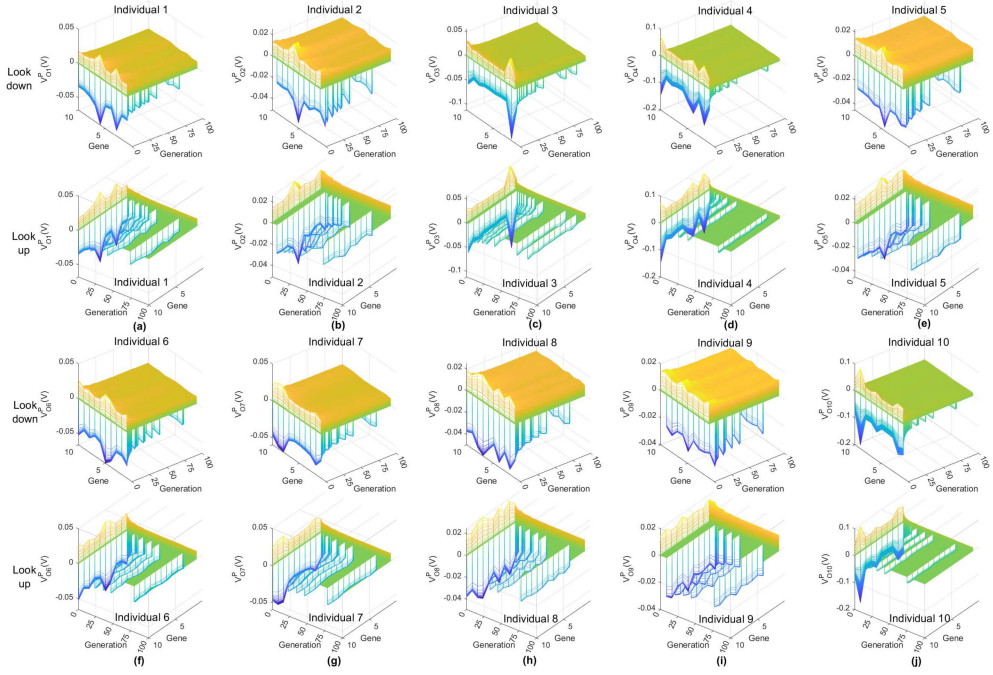

Fig. 12. The simulation results of the parents $V^{\mathrm{P}}_{\mathrm{O}ij}$ in the *parents&update* module over 100 generations for benchmark problem ②. Subfigures (a) through (j) represent the evolution processes in 100 generations from the first parent individual to the 10th parent individual in turn. Each subfigure shows the evolution processes of 10 genes in corresponding individuals over 100 generations. As for each gene, the duration of the whole evolutionary process is 2.0e-5s and the duration of each generation is 2.0e-7s. Each generation has two stages of reading and writing. The first stage is the read operation (duration is 1.0e-7s), as shown in the subfigure where the vertical axis is positive, which is used to produce the corresponding genes in parent individuals. The second stage is the write operation (duration is 1.0e-7s), as shown in the subfigure where the vertical axis is negative, which is used to complete operations similar to selection by changing the memristance of the corresponding memristor in the memristive array. In the writing stage, it can be seen that approximate computing operations are performed only at non-zero $V^{\mathrm{P}}_{\mathrm{O}ij}$ values, which are sparse.

need to accurately write to the currently required optimal value, but to change the corresponding memristances of memristors in a more optimal direction through analog computation. In this way, the computing processes in the write operations are sparse and approximate, so as to fundamentally improve the computing efficiency. Moreover, as shown in Figure 10, the applied voltages in the first stage ($V^{\mathrm{P}}_{\mathrm{r}}$) are less than the thresholds of the memristors in the memristive array of the *parents&update* module, which will not affect the memristance of the memristor when generating information such as offspring. The applied voltages in the second stage ($V^{\mathrm{P}}_{\mathrm{w}i}$) are more than the thresholds of the memristors in memristive array of the *parents&update* module, so they will accordingly change the memristances of memristors in the memristive array.

Based on the aforementioned implementation methods and the preceding simulation experiments of our evolvable brain-inspired hardware system, it can be seen that our design leverages the massively parallel dot-product computation capability of the memristive array [1, 16]. This computing capability of the memristive arrays originates from Ohm's law (i.e., U (Voltage) $\times$ G (Conductance) = I (Current)). In our design, each generation can be completed within one clock

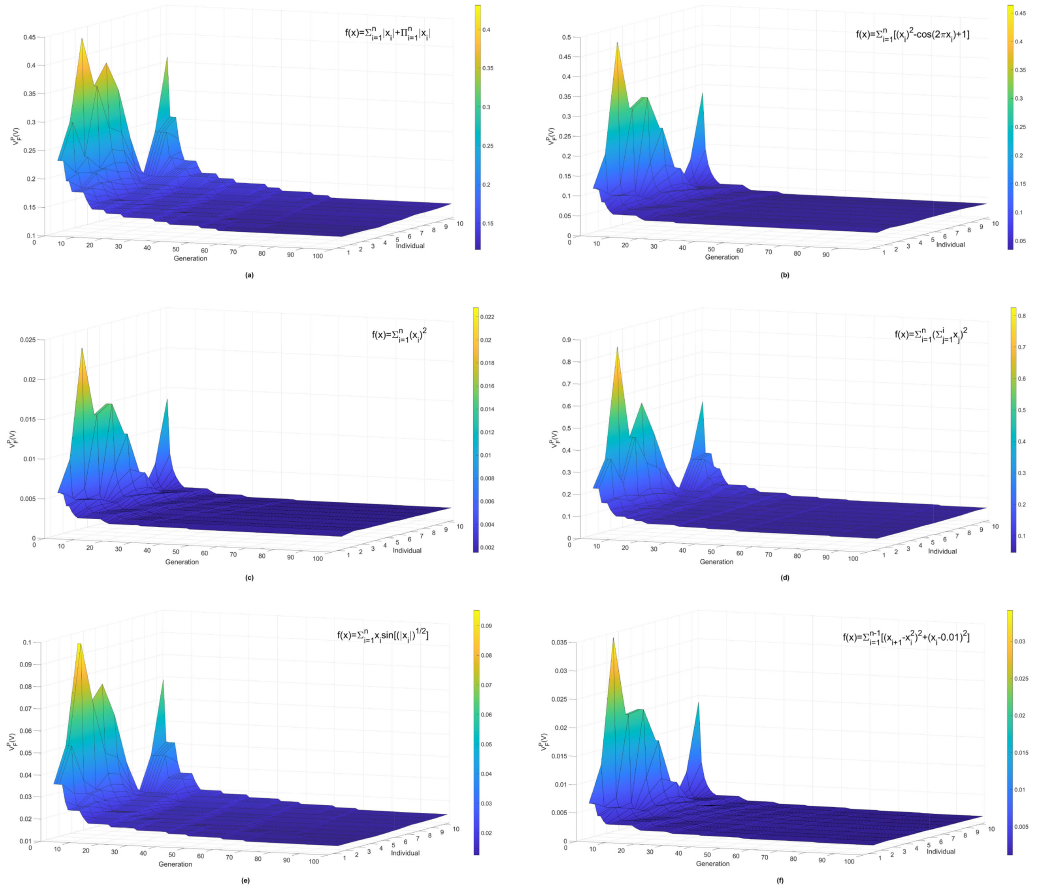

Fig. 13. The simulation results of the evolutionary processes of the fitness for benchmark problems ① through ⑥ over 100 generations (there are 10 individuals in each generation). It can be seen that with the evolution of the population, the fitness of each individual continues to develop toward a required better direction.

cycle. Therefore, in the first half of each clock cycle, we perform reading operations based on Ohm's law by applying the corresponding reading voltages (less than the threshold voltage of the memristor) to the memristive array. The memristive values (representing genes information in the population) are read in the form of currents through parallel analog computing, and these currents are converted into voltages for transmission using the corresponding peripheral circuits (composed of *Adder*, *Subtractor*, *Comparator*, and other modules), allowing the completion of the population reading operations within a single clock cycle. In the second half of each clock cycle, we take advantage of the continuous variability of memristance by applying the corresponding writing voltages (greater than the threshold voltage of the memristor) to the memristive array. This allows us to change the memristances of the corresponding memristors through parallel analog computing, which is also essentially based on Ohm's law, so as to complete the population writing operations within a single clock cycle.

Therefore, during the reading operation, we determine the inferior parent genes, and during the writing operation, we update only these inferior parent genes (hence sparse, as not all parent

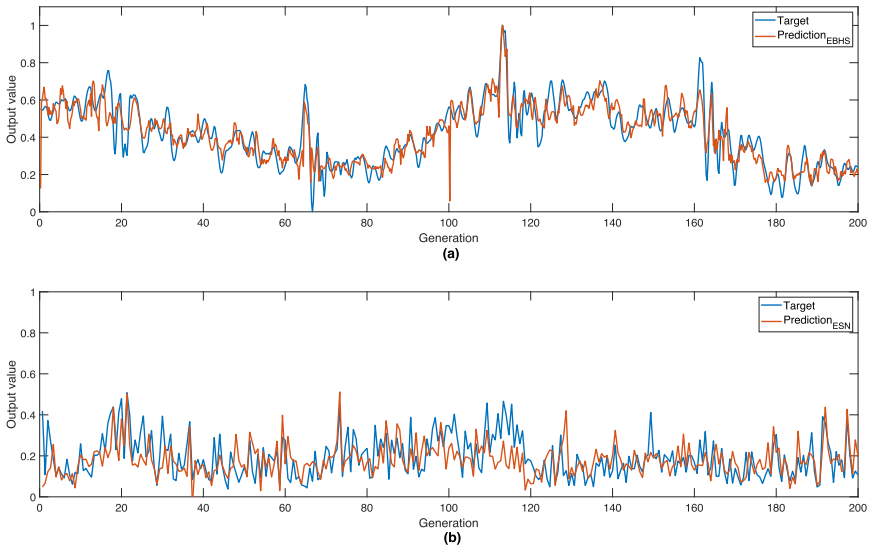

Fig. 14. The experimental results of the prediction problem based on the NARMA20 dataset [51] under different execution ways [52] over 200 generations. (a) The experimental results between the target and the prediction for the memristive neural network optimized by using our evolvable brain-inspired hardware system. (b) The experimental results between target and prediction for the echo state network (ESN) optimized by using traditional evolution programming (a baseline approach).

genes are inferior), enabling them to continuously evolve toward a better direction generation by generation. The update process, which is the writing operation, only requires writing the corresponding genes in a better direction without the need for precisely writing to a specific value (hence, the computing is approximate). In this way, we have effectively combined the high computational power of memristive arrays with the computational rules of evolutionary algorithms, not only greatly improving computational efficiency but also enabling continuous adaptive evolution toward a better direction during the iterative process. The *Adder*, *Subtractor*, *Comparator*, and other modules serve as peripheral circuits to support the computation of the memristive array, thereby assisting in the completion of the aforementioned functions. In general, with the help of the memristive arrays, in situ analog computing operations are performed in parallel, essentially avoiding data transfers between memory and processors. This not only effectively improves computation efficiency but also endows the system with evolutionary capabilities.

Besides, to validate the capability and computational efficiency of the evolvable brain-inspired hardware system we have implemented based on our proposed memristive hardware architecture for evolutionary algorithms, we applied it to optimize a memristive neural network and used the optimized network to perform an actual prediction task for time series based on the NARMA20 dataset [51, 52]. For the prediction task, especially the task of time series prediction [51], an **Echo State Network (ESN)** is typically used as a baseline approach [53]. To make a fair comparison, the ESN is also optimized using the same evolutionary algorithm on the Python software platform. Additionally, the optimized ESN has also performed the same prediction task on the same software platform, serving as a baseline method for comparison with our approach. According to the simulation results as shown in Figure 14 and Table 5, it can be seen that compared to the aforementioned baseline, the network optimized using our proposed evolvable brain-inspired hardware system can efficiently complete optimization tasks. This indicates that our proposed

Table 5. Prediction Accuracy and Execution Time with Different Execution Ways for the Same Prediction Problem Based on the NARMA20 Dataset

Execution Way	*Prediction Accuracy*[†]	*Execution Time*
$Traditional\ software^{\star}$	90.53%	89.9 s
$Memristive\ hardware^{*}$	92.57%	2e-3 s

$\star$ The execution way of traditional software is the echo state network (ESN, a baseline approach) optimized by using traditional evolution programming, and the working frequency of it is 3.8 GHz. $*$ The execution way of memristive hardware is the memristive neural network (MNN) optimized by using our evolvable brain-inspired hardware system, and the working frequency of it is 100 kHz. † The prediction accuracy is calculated through "$Accuracy = 1 - \sqrt{< |y(\hat{t}) - y(t)|^2 >}$", where $y(t)$ is the desired output (target), $y(\hat{t})$ is the actual predicted output of the network, $||$ denotes the Euclidean norm, and $<>$ denotes the empirical mean.

system not only possesses good evolvability for corresponding problems, but also has certain advantages in terms of computational efficiency due to the parallel analog computing with sparsity and approximation.

Moreover, for the benchmark problems $f_{min}(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|$ and $f_{min}(x) = \sum_{i=1}^{n} [(x_i)^2 - cos(2\pi x_i) + 1]$, the fault tolerance experiments are simulated by using the evolvable brain-inspired hardware system. In other words, the memristive array in the *parents&update* module of Figure 3 is set to have a certain probability of damage, whereby the damaged memristor will have its memristance stay at its initial value without plasticity so as to verify the adaptability and fault tolerance of the proposed brain-inspired hardware architecture. For the memristive array used to represent the parents population in Figure 3, memristors are randomly selected for fault settings in each row of the memristive array—that is, each individual in the parents population will have corresponding genes damage randomly. The simulation results of the fault tolerance experiments are shown in Figures 15 through 18 for the evolutionary processes and the fitness of the population in 100 generations. According to the simulation results, when compared to the evolution results of the system without hardware damage, it can be observed that the evolution results of the system with a 50% damage rate have an average decline of approximately 35% in solution quality. So the decline rate of the evolution result is lower than the failure rate of the memristive array, which shows that the evolvable brain-inspired hardware system implemented based on the proposed hardware architecture can realize adaptive evolution based on the current input information in real time. In other words, it can always evolve toward a better direction due to the feature of learning on-the-fly based on spiking signals in the proposed hardware architecture. Therefore, our system can screen out disadvantaged individuals in the population and optimize them in a good direction in the next iteration process, instead of uncontrollably evolving in a worse direction due to an increase in the faulty rate.

On this basis, for the benchmark problem $f_{min}(x) = \sum_{i=1}^{n} [(x_i)^2 - cos(2\pi x_i) + 1]$, the simulation experiments with different kinds of parameter settings for the memristive array (as shown in Figure 19) in the *parents&update* module of Figure 3 are carried out by using the evolvable brain-inspired hardware system. The simulation results are shown in Figure 20. Based on the experimental results in Figure 20(a) and (b), it can be seen that the faster the switching speed of the memristive array is, the larger the evolution step of each generation is, so the system can evolve in a better direction faster, and otherwise it will be slower. Based on the experimental results in Figure 20(c) and (d), it can be observed that the larger the range of high and low memristance

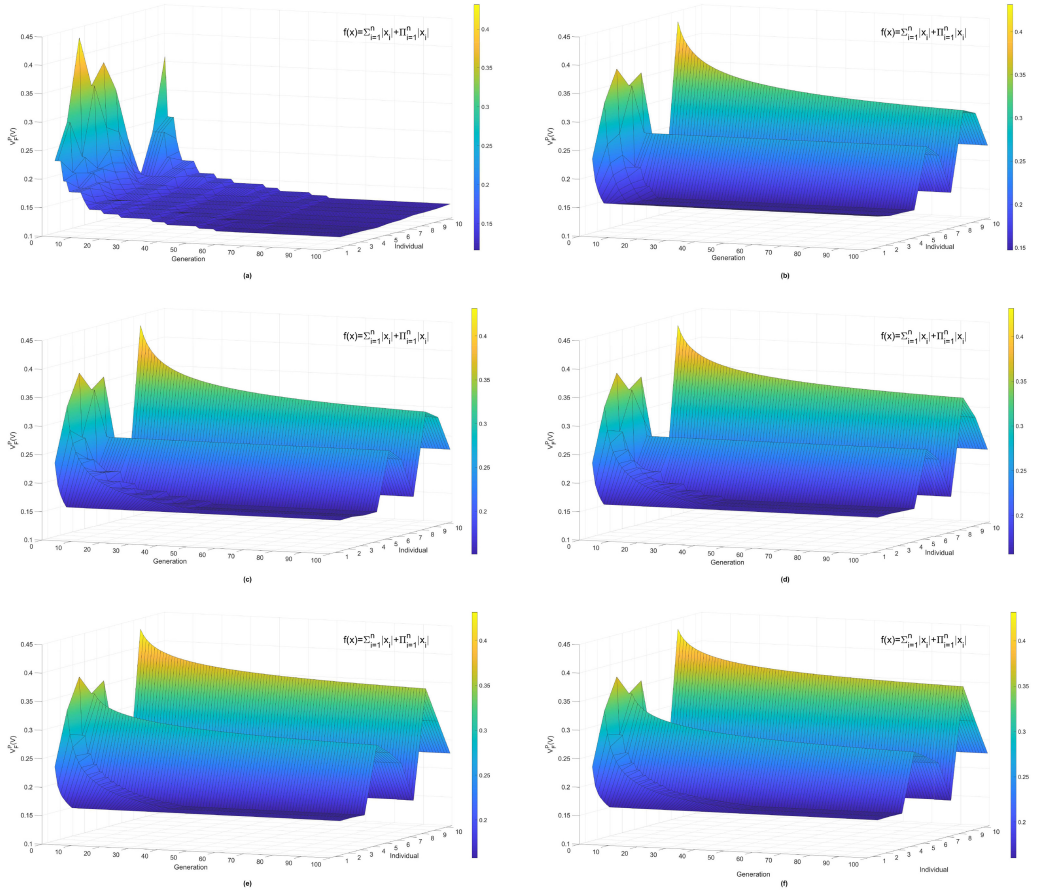

Fig. 15. The fault tolerance experiment results for benchmark problem ① with different failure rates in the memristive array. It can be seen that with the evolution of the population, there will always be individual whose fitness will continue to develop toward a better direction even if there are faults in the circuit. (a) The simulation results of the evolutionary processes of the fitness over 100 generations under a 0% failure rate of the memristive array. (b) The simulation results of the evolutionary processes of the fitness over 100 generations under a 10% failure rate of the memristive array. (c) The simulation results of the evolutionary processes of the fitness over 100 generations under a 20% failure rate of the memristive array. (d) The simulation results of the evolutionary processes of the fitness over 100 generations under a 30% failure rate of the memristive array. (e) The simulation results of the evolutionary processes of the fitness over 100 generations under a 40% failure rate of the memristive array. (f) The simulation results of the evolutionary processes of the fitness over 100 generations under a 50% failure rate of the memristive array.

states, the broader the solution space for genes within each individual in the population, so the potential for the system to find better solutions will be greater.

5.2 Performance Analysis and Comparison for the Proposed Brain-Inspired Hardware Architecture

As for the same benchmark problem in Table 4, the simulation results based on the evolvable brain-inspired hardware system (executed on the PSPICE simulation platform) are respectively compared

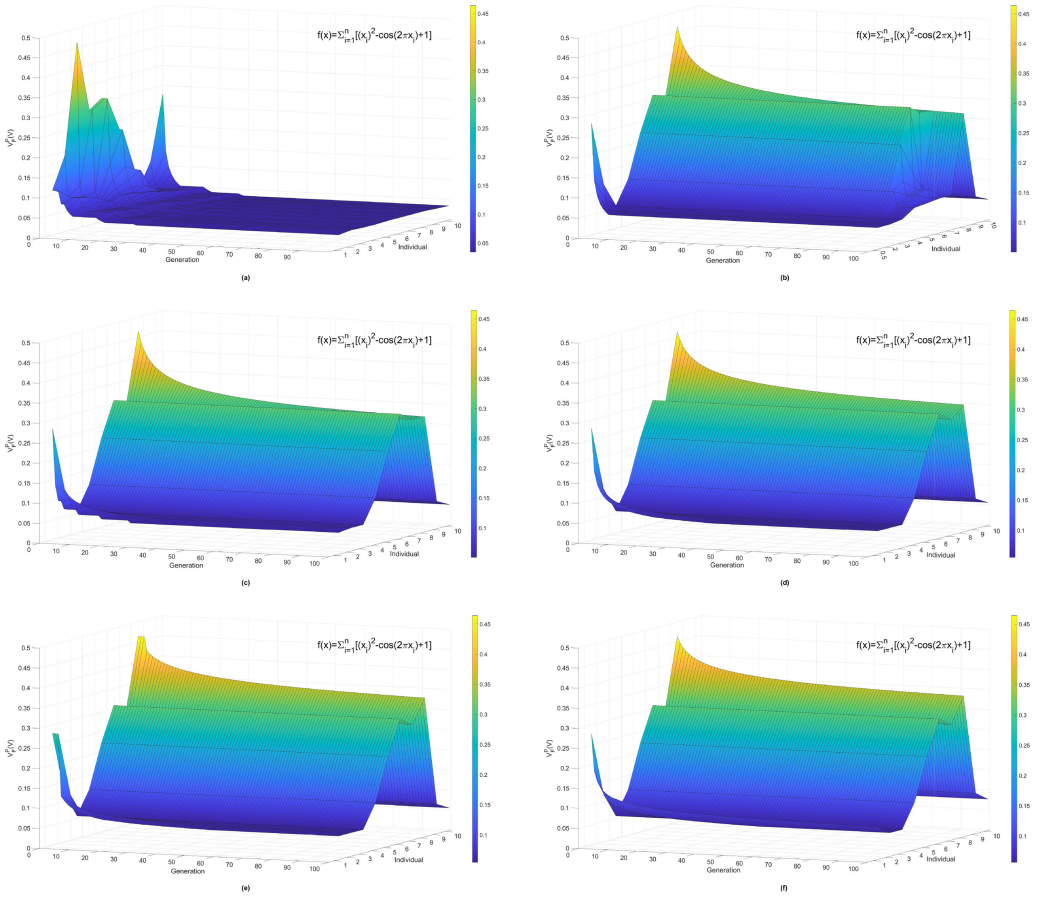

Fig. 16. The fault tolerance experiment results for benchmark problem ② with different failure rates in the memristive array. It can be seen that with the evolution of the population, there will always be an individual whose fitness will continue to develop toward a better direction even if there are faults in the circuit. (a) The simulation results of the evolutionary processes of the fitness over 100 generations under a 0% failure rate of the memristive array. (b) The simulation results of the evolutionary processes of the fitness over 100 generations under a 10% failure rate of the memristive array. (c) The simulation results of the evolutionary processes of the fitness over 100 generations under a 20% failure rate of the memristive array. (d) The simulation results of the evolutionary processes of the fitness over 100 generations under a 30% failure rate of the memristive array. (e) The simulation results of the evolutionary processes of the fitness over 100 generations under a 40% failure rate of the memristive array. (f) The simulation results of the evolutionary processes of the fitness over 100 generations under a 50% failure rate of the memristive array.

with those based on the traditional software system with evolutionary algorithms (executed on the MATLAB simulation platform), whereby the following features can be observed.

In terms of iteration speed of the evolutionary process, it can be seen that the evolvable brain-inspired hardware system implemented based on the proposed hardware architecture has a speed advantage as shown in Table 6. The reasons for the speed advantage can be analyzed from the following two aspects. The first is that hardware computing itself has the advantage of running speed different from software computing. Specifically, the working frequency of the evolvable brain-inspired hardware system essentially depends on the read/write speed of the memristors,

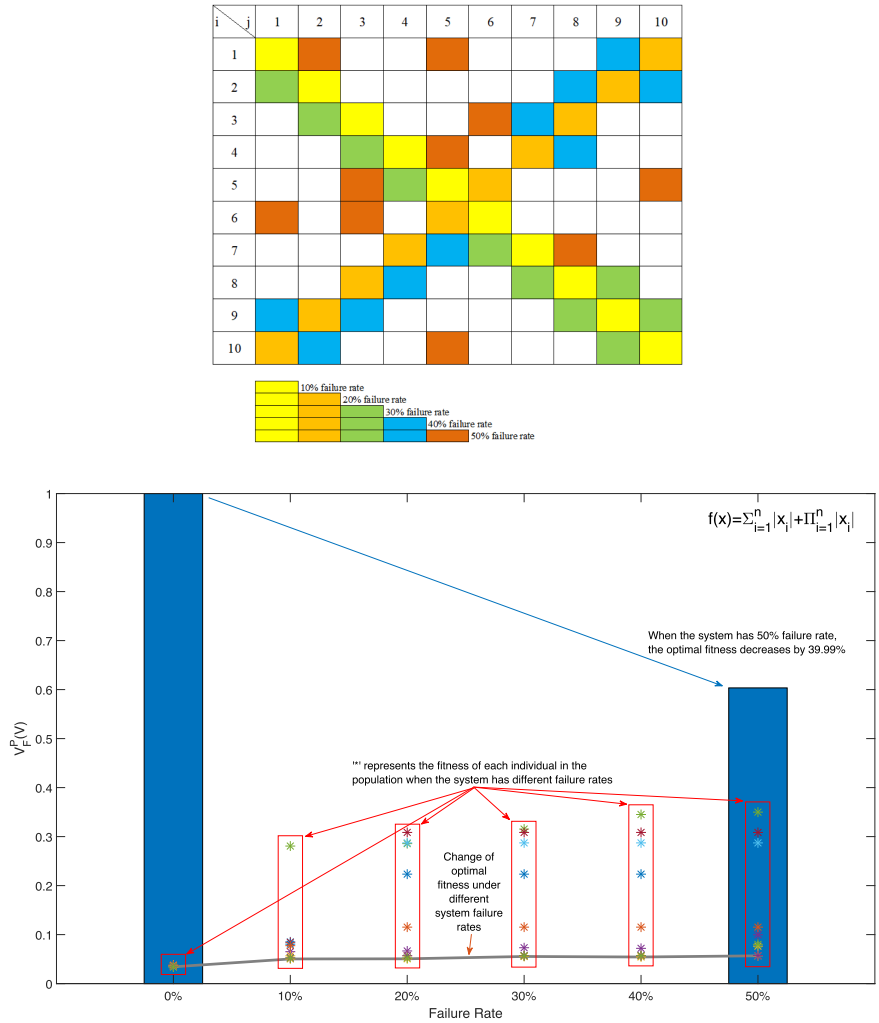

Fig. 17. The statistics of the fault tolerance experiments with the evolvable brain-inspired hardware system under different failure rates of memristive array for benchmark problem ①. The top subfigure shows the damage degree and distribution of the memristive array in the hardware system. The bottom subfigure shows the evolved results when the system is at 0% and 10% to 50% failure rates, respectively (as indicated in the red boxes). In this benchmark problem, the lower the fitness value, the better the evolution result. On this basis, we compared the optimal evolutionary result of the system with a 50% hardware damage rate to the optimal evolutionary result of the undamaged system and calculated the decrease rate of the evolutionary results (as shown by the two blue bars).

and the read/write speed of the memristive devices that have been reported so far can reach the level of 1e-10s to 1e-12s [54]. In this way, the whole evolution processes can be set to different working frequencies based on the population sizes and iteration times under the premise of meeting the read/write speed of memristor. As shown in Table 6, the working frequencies of the evolvable brain-inspired hardware system in the simulation experiments are set to 5 MHz, which meets the read/write speed of the memristor mentioned earlier, and the results show that the corresponding

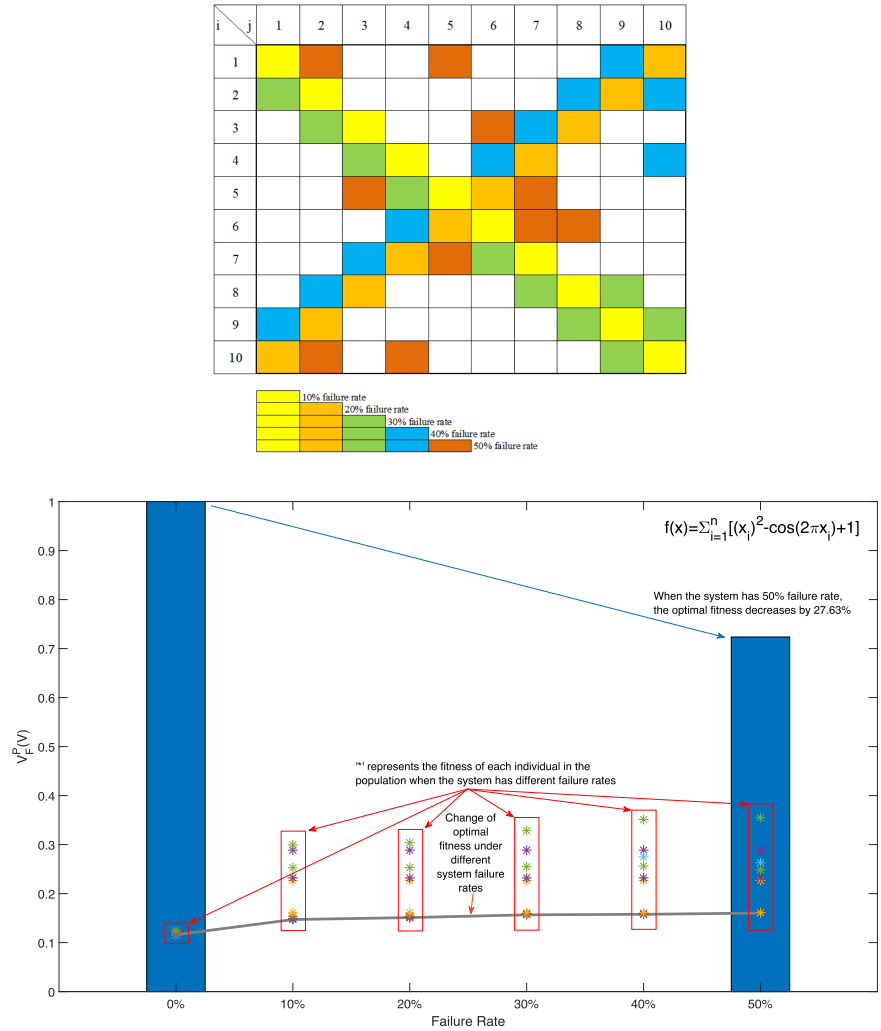

Fig. 18. The statistics of the fault tolerance experiments with the evolvable brain-inspired hardware system under different failure rates of the memristive array for benchmark problem ②. The top subfigure shows the damage degree and distribution of the memristive array in the hardware system. The bottom subfigure shows the evolved results when the system is at 0% and 10% to 50% failure rates, respectively (as indicated in the red boxes). In this benchmark problem, the lower the fitness value, the better the evolution result. On this basis, we compared the optimal evolutionary result of the system with a 50% hardware damage rate to the optimal evolutionary result of the undamaged system and calculated the decrease rate of the evolutionary results (as shown by the two blue bars).

benchmark problem can be completed within the set frequency. The second is that this evolvable brain-inspired hardware system is implemented based on a brain-inspired hardware architecture, whereby the memristive arrays in this architecture are used as core computing units to realize brain-inspired analog computing in parallel. In this way, computations in the evolutionary processes are sparse and approximate, which can effectively improve the computing efficiency compared with the dense and accurate computing in the traditional software system. Exactly as shown

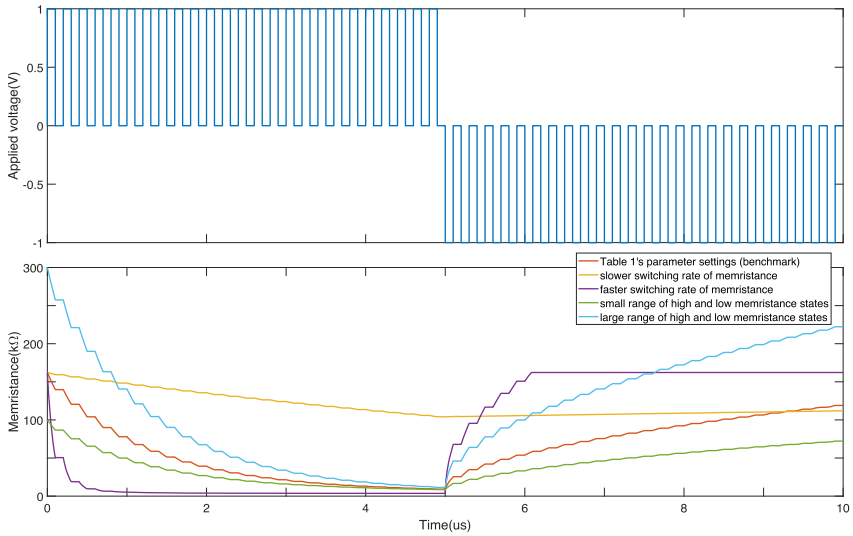

Fig. 19. Memristance change curves of the DSAM model based on different parameter settings, which are used to carry out corresponding experiments in Figure 20 for benchmark problem ②.

Table 6. Execution Time with Different Population Sizes and Different Execution Ways for the Same Benchmark Problem

Population Sizes	*Traditional Software**	*Memristive Hardware**
5 ×5	0.167s	2e-5s
10 ×10	0.398s	2e-5s
100 ×100	28.599s	2e-5s

* The working frequency of software-based implementation is 2.5 GHz, and parallel computing is adopted during program execution. * The working frequency of hardware-based implementation is 5 MHz.

in Table 6, the evolvable brain-inspired hardware system can complete the evolution faster at lower working frequency.

In terms of adaptability and fault tolerance, the simulation results of the fault tolerance experiments and the experiments with different parameter settings in Figures 15 through 20 show that the evolvable brain-inspired hardware system can screen out the disadvantaged individuals in the population and optimize them in a good direction in the process of next iteration, instead of uncontrollably evolving in a worse direction due to an increase in faulty rate or different parameter settings. In this way, the evolvable system implemented based on the proposed brain-inspired hardware architecture for evolutionary algorithms based on memristive arrays can have the ability of adaptive optimization based on real-time environmental information by an on-the-fly learning way—that is, have good adaptability and fault tolerance.

In terms of hardware overhead of the implemented functional hardware system, as can be seen from the circuit modules of the bottom layer shown in Figures 5 through 7, our system mainly consists of three basic two-terminal components: memristors, resistors, and transistors. These underlying circuit units are used to perform the corresponding computational operations. Based on this foundation, and following a bottom-up circuit design approach, the corresponding unit circuits

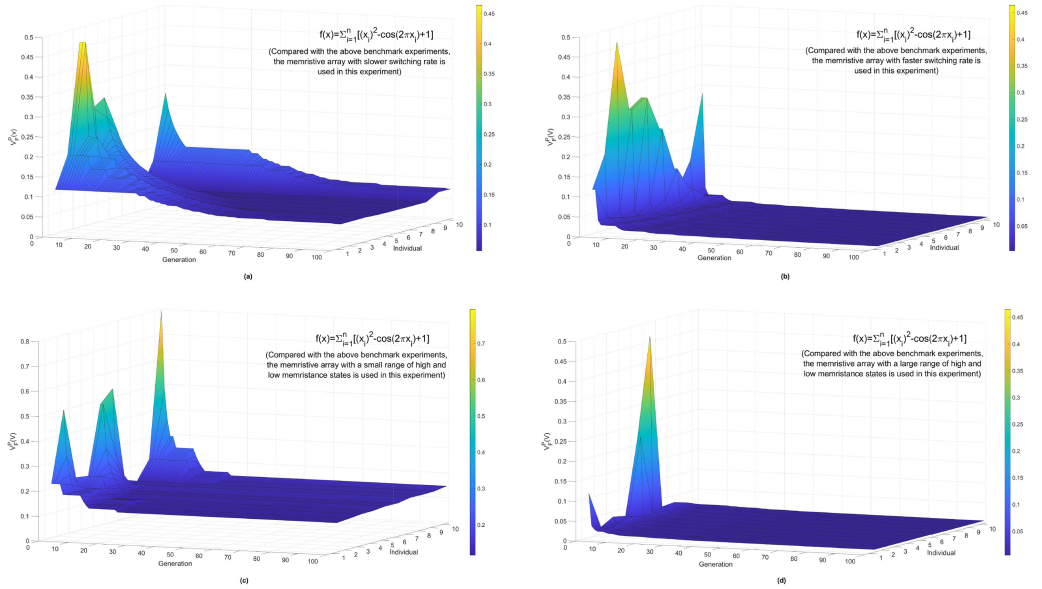

Fig. 20. The simulation results of the evolutionary processes of the fitness for benchmark problem ② over 100 generations (there are 10 individuals in each generation), which are based on different kinds of parameter settings for the memristive array as shown in Figure 19. (a) The experimental setting of the memristive array is in a slower switching speed, and it can be seen that their fitness evolves slowly in a better direction. (b) The experimental setting of the memristive array is in a faster switching speed, and it can be seen that their fitness evolves faster in a better direction. (c) The experimental setting of the memristive array is in a small range of high and low memristance states. (d) The experimental setting of the memristive array is in a large range of high and low memristance states. Based on the results of (c) and (d), it can be observed that the larger the range of high and low memristance states, the broader the solution space for genes within each individual in the population, so the potential for the system to find better solutions will be greater.

are combined and cascaded according to the computational rules of the evolutionary algorithm to implement intermediate functional modules. Then, drawing inspiration from the connection structure of biological neural networks, the relevant functional modules are cascaded to create an evolvable hardware system. Therefore, we estimated the hardware overhead of a system with a scale of $m \cdot n$ by counting the number of these three types of components [52], and based on their sizes, we made a rough estimation of the order of magnitude for the circuit area. So as for a brain-inspired hardware architecture for evolutionary algorithms with a scale of $m \cdot n$, their hardware overhead is shown in Table 7. In the *fitness calculation&selection* module, since different goals (problems) will correspond to different fitness calculation units, only the hardware overhead of the section used to complete the *selection* operation (as shown in Figure 3) is counted. It can be seen that the increase of hardware overhead is linear with the scale of hardware architecture ($m \cdot n$). The relevant study has shown that the scale of devices such as memristors, transistors, and resistors can reach the level of several to dozens of nanometers [18], so based on the number of these three types of components listed in Table 7, we can calculate that to construct a system with a scale of $m \cdot n$, approximately $184 \cdot m \cdot n$ two-terminal components (including memristors, resistors, and transistors) are needed. Assuming that each two-terminal component has a size 10 nm (nanometers), the area of one component would be about 10 nm^2. Therefore, $184 \cdot m \cdot n \times 100\ nm^2 = 0.184 \cdot m \cdot n\ (\mu m^2)$ (square microns), so we can see that the area of a system with a scale of $m \cdot n$

Table 7. Hardware Overhead of the Evolvable Brain-Inspired Hardware System Implemented Based on the Proposed Brain-Inspired Hardware Architecture for Evolutionary Algorithms

	Parents&update Module	*Mutation&offspring Module*	*Selection Module*
Memristor	$m \cdot n$	$3 \cdot m \cdot n$	0
Transistor	$8 \cdot m \cdot n$	$69 \cdot m \cdot n$	$33 \cdot m$
Resistor	$8 \cdot m \cdot n$	$54 \cdot m \cdot n$	$8 \cdot m$
Order of Magnitude for Hardware Overhead		$(m \cdot n)\ \mu m^2$	

Table 8. Power Consumption of the Evolvable Brain-Inspired Hardware System Implemented Based on the Proposed Brain-Inspired Hardware Architecture for Evolutionary Algorithms

	Parents&update Module	*Mutation&offspring Module*	*Selection Module*
Memristor	7.5uW·$(m \cdot n)$	10nW·$(3 \cdot m \cdot n)$	0
Transistor	20uW·$(8 \cdot m \cdot n)$	20uW·$(64 \cdot m \cdot n)$+5uW·$(5 \cdot m \cdot n)$	20uW·$(8 \cdot m)$+5uW·$(25 \cdot m)$
Resistor	2.5nW·$(8 \cdot m \cdot n)$	2.5nW·$(54 \cdot m \cdot n)$	2.5nW·$(8 \cdot m)$
Order of Magnitude		1.75·$(m \cdot n)$mW	

is roughly $0.2 \cdot m \cdot n\ \mu m^2$. However, for a real circuit system, its area is not only related to the size and number of components but also to the layout of components within the circuit and their interconnections. Hence, we round up to provide a more conservative estimation, which is $m \cdot n\ \mu m^2$. In this way, as the scale of the hardware architecture grows $(m \cdot n)$, the hardware overhead of the proposed brain-inspired architecture for evolutionary algorithms grows in a controllable linear fashion so that within a certain scale of hardware architecture, the computational efficiency will not be compromised.

On this basis, according to the power measurement data of the PSPICE simulation platform, the power consumption of the proposed brain-inspired hardware architecture for evolutionary algorithms is estimated as follows. The average power consumption of memristors in the *parents&update* module is approximately 7.5 uW per memristor. The average power consumption of memristors in the *mutation&offspring* module is approximately 10 nW per memristor. The average power consumption of transistors used to realize addition, subtraction, proportion, and other calculation functions is approximately 20 uW per transistor. The average power consumption of transistors used to implement the comparator, switch, and so forth is approximately 5 uW per transistor. The average current flowing through the resistor is approximately 0.5 uA, and taking a 10k Ω resistor as an example, the average power consumption of it is approximately 2.5 nW per resistor. Therefore, as shown in Table 8, the order of magnitude for the average power consumption of the proposed brain-inspired hardware architecture for evolutionary algorithms with a scale of $m \cdot n$ can be considered as 1.75 mW. In actual operation, due to the brain-inspired computing (sparse, approximate, and parallel) capacity, the actual power consumption of this hardware architecture will be lower theoretically.

In addition, as shown in Table 9, compared with the work in FPGA-based hardware implementation for evolutionary algorithms [38, 55], the proposed brain-inspired hardware architecture adopts a brain-inspired update mechanism to realize efficiently sparse and approximate computing in the form of analog circuit, which has a certain improvement in execution speed.

Table 9. Comparisons Between the Memristor-Based Hardware Implementation for Evolutionary Algorithms to the FPGA-Based Hardware Implementation

	Memristor Based	FPGA Based [38, 55]
Brain-Inspired Update Mechanism	Yes	No
Sparse and Approximate Computing	Yes	No
Analog Circuit Implementation	Yes	No
Order of Magnitude of Execution Speed	*us∼ms*	*ms∼s*

In general, considering the aforementioned implementation methods and the comparative analysis of our evolvable brain-inspired hardware system, if in practical applications we need to extend our system to larger populations, number of genes, number of generations, and so forth based on the problem to be solved, we can accomplish it using the following strategies.

First, for expanding the size of the population and the number of genes, the main task is to increase the size of the proposed system's memristive arrays. As shown in Figure 3, in our design, the population is mapped to the memristive array, with each memristor representing a gene, a row of memristors representing an individual, and many rows of memristors forming a population. Meanwhile, the corresponding peripheral circuits (the blue, green areas, etc., in Figure 3), which assist the memristive arrays in efficiently performing the computational operations of the evolutionary algorithm in our proposed system, also need to be correspondingly increased. Of course, this will undoubtedly lead to an increase in overall power consumption, hardware overhead, or other metrics. However, it brings about high computational efficiency, adaptive evolution, and good fault tolerance capabilities. And based on our analysis in Tables 7 and 8, the hardware overhead and power consumption increase approximately linearly with the expansion of population size, so they are controllable.

Second, for increasing the number of generations, the main approach is to either increase the working frequency of the proposed system or extend the system's working (running) time. As shown in Figures 10 through 12, one generation of population evolution can be completed within one clock cycle, so the faster the working frequency, the more generations of population evolution can be iterated within the same amount of working (running) time in our design. For instance, if we need to increase the number of generations from 100 to 200, we can either double the system's working frequency without changing the working (running) time or double the working (running) time while maintaining the original working frequency. The former approach requires our system's core computing components (i.e., memristors) to have the corresponding reading/writing speed. Current research shows that the reading/writing speed of memristors can reach 1e-12s [54]. Therefore, the system's operating frequency can be increased by about five orders of magnitude based on the 5MHz frequency used in our work. The latter approach does not require any changes to the system and only needs to extend the operating time.

Furthermore, as the scale of our implemented evolvable brain-inspired hardware system expands, due to the use of memristive arrays as the core computational module, they inevitably require continuous reading and writing operations to ensure that the system performs the corresponding computational tasks according to our requirements. This may bring about associated reprogramming overhead issues when frequently performing reading and writing operations on the memristive arrays. However, our design actually helps alleviate these issues. First of all, the concept of sparsity was incorporated into our design—that is, as for the population (i.e., the $m \cdot n$ memristive array in Figure 3), only the non-optimal genes in parents (the corresponding memristors in the memristive array) need to be updated to make them change toward a required better

direction. In this way, the computing processes are sparse, so as to effectively improve the computing efficiency—that is, the writing processes are sparse, which will reduce the reprogramming overhead for the memristive weights. Then, for the operations of updating non-optimal genes (i.e., to change the memristances of the corresponding memristors in the memristive array), it is not necessary to accurately change the corresponding genes (i.e., the corresponding memristances in the memristive array) to the optimal values but to change toward a better direction so that the updating operations are approximate. Therefore, in our design, the memristive arrays as the core computational module require continuous reading and writing operations, but due to the presence of sparsity and approximation, the reading and writing operations are different from traditional ones, which will help alleviate the overheads to reprogram memristor weights. As shown in Figure 10, reading operations are performed in the first half of one clock cycle and writing operations in the second half. These operations can be smoothly completed according to the pulse sequence. Consequently, on the whole, the overheads to reprogram memristor weights will not pose an uncontrollable impact on our system.

6 CONCLUSION

In this work, a brain-inspired hardware architecture for evolutionary algorithms is proposed based on memristive arrays. This architecture can realize efficient brain-inspired computing by implementing the corresponding evolvable brain-inspired hardware system, drawing on the computing principles of evolutionary algorithms and the structure of biological neural networks. The proposed hardware architecture is mainly composed of three modules: the *parents&update* module, the *mutation&offspring* module, and the *fitness calculation&selection* module. They cooperate with each other to form an efficient computational flow, enabling adaptive evolution through brain-inspired computing.

Memristors are used as the core computing elements in the memristive arrays of the proposed hardware architecture. As an emerging electronic component, memristors can perform computation via physical laws at the same location where information is stored. In this way, the proposed hardware architecture can minimize the need for data transfer between the memory and processor, enabling in-memory analog computing to speed up the computing processes of the system. Simultaneously, by referencing the computing principles of evolutionary algorithms and the structure of biological neural networks, the proposed hardware system can achieve sparse and approximate computing in the evolutionary processes. This approach closely mimics the processing mechanism of the brain, fundamentally improving the system's working efficiency. Furthermore, the proposed hardware architecture operates in an online training mode, continuously evolving toward better performance, ensuring that the hardware system implemented based on it has high stability and fault tolerance.

In the future, we will further optimize the performance of the brain-inspired hardware architecture and improve the implementation method of the memristive hardware system, so as to realize a larger-scale reconfigurable memristive hardware system to solve more practical learning and optimization problems, and explore potential association between biological systems and brain-inspired memristive hardware systems.

REFERENCES

[1] Yang Zhang, Zhongrui Wang, Jiadi Zhu, Yuchao Yang, Mingyi Rao, Wenhao Song, Ye Zhuo, Xumeng Zhang, Menglin Cui, and Linlin Shen. 2020. Brain-inspired computing with memristors: Challenges in devices, circuits, and systems. *Applied Physics Reviews* 7, 1 (2020), 011308. https://aip.scitation.org/doi/full/10.1063/1.5124027.

[2] Yoeri Van De Burgt and Paschalis Gkoupidenis. 2020. Organic materials and devices for brain-inspired computing: From artificial implementation to biophysical realism. *MRS Bulletin* 45, 8 (2020), 631–640.

[3] Yesheng Li and Kah-Wee Ang. 2021. Hardware implementation of neuromorphic computing using large-scale memristor crossbar arrays. *Advanced Intelligent Systems* 3, 1 (2021), 2000137.
[4] Jianshi Tang, Fang Yuan, Xinke Shen, Zhongrui Wang, Mingyi Rao, Yuanyuan He, Yuhao Sun, Xinyi Li, Wenbin Zhang, and Yijun Li. 2019. Bridging biological and artificial neural networks with emerging neuromorphic devices: Fundamentals, progress, and challenges. *Advanced Materials* 31, 49 (2019), 1902761.
[5] Qiangfei Xia and J. Joshua Yang. 2019. Memristive crossbar arrays for brain-inspired computing. *Nature Materials* 18, 4 (2019), 309–323.
[6] Mark Horowitz. 2014. 1.1 Computing's energy problem (and what we can do about it). In *Proceedings of the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, Los Alamitos, CA, 10–14.
[7] Daniele Ielmini and H.-S. Philip Wong. 2018. In-memory computing with resistive switching devices. *Nature Electronics* 1, 6 (2018), 333–343.
[8] Teng Zhang, Ke Yang, Xiaoyan Xu, Yimao Cai, Yuchao Yang, and Ru Huang. 2019. Memristive devices and networks for brain-inspired computing. *Physica Status Solidi (RRL)–Rapid Research Letters* 13, 8 (2019), 1900029.
[9] Leon Chua. 1971. Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory* 18, 5 (1971), 507–519.
[10] Dmitri B. Strukov, Gregory S. Snider, Duncan R. Stewart, and R. Stanley Williams. 2008. The missing memristor found. *Nature* 453, 7191 (2008), 80–83.
[11] Zilu Wang and Xiaoping Wang. 2017. A novel memristor-based circuit implementation of full-function Pavlov associative memory accorded with biological feature. *IEEE Transactions on Circuits and Systems I: Regular Papers* 65, 7 (2017), 2210–2220.
[12] Junwei Sun, Gaoyong Han, Zhigang Zeng, and Yanfeng Wang. 2019. Memristor-based neural network circuit of full-function Pavlov associative memory with time delay and variable learning rate. *IEEE Transactions on Cybernetics* 50, 7 (2019), 2935–2945.
[13] Can Li, Zhongrui Wang, Mingyi Rao, Daniel Belkin, Wenhao Song, Hao Jiang, Peng Yan, Yunning Li, Peng Lin, and Miao Hu. 2019. Long short-term memory networks in memristor crossbar arrays. *Nature Machine Intelligence* 1, 1 (2019), 49–57.
[14] Pilin Junsangsri and Fabrizio Lombardi. 2012. Design of a hybrid memory cell using memristance and ambipolarity. *IEEE Transactions on Nanotechnology* 12, 1 (2012), 71–80.
[15] Shimeng Yu. 2018. Neuro-inspired computing with emerging nonvolatile memorys. *Proceedings of the IEEE* 106, 2 (2018), 260–285.
[16] Yibo Li, Zhongrui Wang, Rivu Midya, Qiangfei Xia, and J. Joshua Yang. 2018. Review of memristor devices in neuromorphic computing: Materials sciences and device challenges. *Journal of Physics D: Applied Physics* 51, 50 (2018), 503002.
[17] Jaeyoung Park. 2020. Neuromorphic computing using emerging synaptic devices: A retrospective summary and an outlook. *Electronics* 9, 9 (2020), 1414.
[18] Shuang Pi, Can Li, Hao Jiang, Weiwei Xia, Huolin Xin, J. Joshua Yang, and Qiangfei Xia. 2019. Memristor crossbar arrays with 6-nm half-pitch and 2-nm critical dimension. *Nature Nanotechnology* 14, 1 (2019), 35–39.
[19] Muhammad Ismail, Umesh Chand, Chandreswar Mahata, Jamel Nebhen, and Sungjun Kim. 2022. Demonstration of synaptic and resistive switching characteristics in W/TiO2/HfO2/TaN memristor crossbar array for bioinspired neuromorphic computing. *Journal of Materials Science & Technology* 96 (2022), 94–102.
[20] Jalil Boukhobza, Stéphane Rubini, Renhai Chen, and Zili Shao. 2017. Emerging NVM: A survey on architectural integration and research challenges. *ACM Transactions on Design Automation of Electronic Systems* 23, 2 (2017), 1–32.
[21] Jia Chen, Jiancong Li, Yi Li, and Xiangshui Miao. 2021. Multiply accumulate operations in memristor crossbar arrays for analog computing. *Journal of Semiconductors* 42, 1 (2021), 013104.
[22] Adam Safron, Victoria Klimaj, and Inês Hipólito. 2022. On the importance of being flexible: Dynamic brain networks and their potential functional significances. *Frontiers in Systems Neuroscience* 15 (2022), 149.
[23] Zongyuan Cai and Xinze Li. 2021. Neuromorphic brain-inspired computing with hybrid neural networks. In *Proceedings of the 2021 IEEE International Conference on Artificial Intelligence and Industrial Design*. IEEE, Los Alamitos, CA, 343–347.
[24] Tetsuya Higuchi, Masaya Iwata, Didier Keymeulen, Hidenori Sakanashi, Masahiro Murakawa, Isamu Kajitani, Eiichi Takahashi, Kenji Toda, N. Salami, and Nobuki Kajihara. 1999. Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation* 3, 3 (1999), 220–235.
[25] Xin Yao and Tetsuya Higuchi. 1999. Promises and challenges of evolvable hardware. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 29, 1 (1999), 87–97.
[26] Jeffrey R. Sampson. 1976. *Adaptation in Natural and Artificial Systems (John H. Holland)*. Society for Industrial and Applied Mathematics.
[27] Thomas Back. 1996. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press.

[28] Falko Dressler and Ozgur B. Akan. 2010. Bio-inspired networking: from theory to practice. *IEEE Communications Magazine* 48, 11 (2010), 176–183.
[29] Paulo Leitão, José Barbosa, and Damien Trentesaux. 2012. Bio-inspired multi-agent systems for reconfigurable manufacturing systems. *Engineering Applications of Artificial Intelligence* 25, 5 (2012), 934–944.
[30] Agoston E. Eiben and Jim E. Smith. 2015. What is an evolutionary algorithm? In *Introduction to Evolutionary Computing*. Springer, 25–48.
[31] Ananda Samajdar, Parth Mannan, Kartikay Garg, and Tushar Krishna. 2018. GeneSys: Enabling continuous learning through neural network evolution in hardware. In *Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. 855–866. DOI: http://dx.doi.org/10.1109/MICRO.2018.00074
[32] Chia-Hua Chuang, Chun-Liang Lin, Yen-Chang Chang, Tanagorn Jennawasin, and Po-Kuei Chen. 2013. Design of synthetic biological logic circuits based on evolutionary algorithm. *IET Systems Biology* 7, 4 (2013), 89–105.
[33] Mohammad Nadji-Tehrani and Ali Eslami. 2020. A brain-inspired framework for evolutionary artificial general intelligence. *IEEE Transactions on Neural Networks and Learning Systems* 31, 12 (2020), 5257–5271.
[34] Stephen D. Scott, Ashok Samal, and Shared Seth. 1995. HGA: A hardware-based genetic algorithm. In *Proceedings of the 1995 ACM 3rd International Symposium on Field-Programmable Gate Arrays*. 53–59.
[35] Jin Jung Kim and Daek Jin Chung. 1999. Implementation of genetic algorithm based on hardware optimization. In *Proceedings of IEEE. IEEE Region 10 Conference. TENCON 99. 'Multimedia Technology for Asia-Pacific Information Infrastructure' (Cat. No. 99CH37030)*, Vol. 2. IEEE, Los Alamitos, CA, 1490–1493.
[36] Norihiko Yoshida and Tomohiro Yasuoka. 1999. Multi-gap: Parallel and distributed genetic algorithms in VLSI. In *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 99CH37028)*, Vol. 5. IEEE, Los Alamitos, CA, 571–576.
[37] Zhenhuan Zhu, David J. Mulvaney, and Vassilios A. Chouliaras. 2007. Hardware implementation of a novel genetic algorithm. *Neurocomputing* 71, 1-3 (2007), 95–106.
[38] Pei-Yin Chen, Ren-Der Chen, Yu-Pin Chang, Leang-San Shieh, and Heidar A. Malki. 2008. Hardware implementation for a genetic algorithm. *IEEE Transactions on Instrumentation and Measurement* 57, 4 (2008), 699–705.
[39] Huihan Li, Shaocong Wang, Xumeng Zhang, Wei Wang, Rui Yang, Zhong Sun, Wanxiang Feng, Peng Lin, Zhongrui Wang, and Linfeng Sun. 2021. Memristive crossbar arrays for storage and computing applications. *Advanced Intelligent Systems* 3, 9 (2021), 2100017.
[40] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J. Joshua Yang, and He Qian. 2020. Fully hardware-implemented memristor convolutional neural network. *Nature* 577, 7792 (2020), 641–646.
[41] Yong-Bin Yu, Chen Zhou, Quan-Xin Deng, Man Cheng, and Zheng-Fei Kang. 2022. Memristor-based genetic algorithm for image restoration. *Journal of Electronic Science and Technology* 20, 2 (2022), 100158.
[42] Zhuojun Chen, Judi Zhang, Shuangchun Wen, Ya Li, and Qinghui Hong. 2021. Competitive neural network circuit based on winner-take-all mechanism and online Hebbian learning rule. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 6 (2021), 1095–1107.
[43] Yi Li, Yingpeng Zhong, Jinjian Zhang, Lei Xu, Qing Wang, Huajun Sun, Hao Tong, Xiaoming Cheng, and Xiangshui Miao. 2014. Activity-dependent synaptic plasticity of a chalcogenide electronic synapse for neuromorphic systems. *Scientific Reports* 4, 1 (2014), 1–7.
[44] Zdeněk Biolek, Dalibor Biolek, and Viera Biolkova. 2009. SPICE model of memristor with nonlinear dopant drift. *Radioengineering* 18, 2 (2009), 210–214.
[45] Charles Darwin. 1909. *The Origin of Species*. PF Collier & Son, New York, NY.
[46] Agoston E. Eiben and Jim Smith. 2015. From evolutionary computation to the evolution of things. *Nature* 521, 7553 (2015), 476–482.
[47] David B. Fogel. 1994. An introduction to simulated evolutionary optimization. *IEEE Transactions on Neural Networks* 5, 1 (1994), 3–14.
[48] Xin Yao, Yong Liu, and Guangming Lin. 1999. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation* 3, 2 (1999), 82–102.
[49] Yunning Li, Wenhao Song, Zhongrui Wang, Hao Jiang, Peng Yan, Peng Lin, Can Li, Mingyi Rao, Mark Barnell, and Qing Wu. 2022. Memristive field-programmable analog arrays for analog computing. *Advanced Materials* 2022 (2022), 2206648.
[50] Peng Zhou, Dong-Uk Choi, Wei D. Lu, Sung-Mo Kang, and Jason K. Eshraghian. 2022. Gradient-based neuromorphic learning on dynamical RRAM arrays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12, 4 (2022), 888–897.
[51] Amir F. Atiya and Alexander G. Parlos. 2000. New results on recurrent network training: Unifying the algorithms and accelerating convergence. *IEEE Transactions on Neural Networks* 11, 3 (2000), 697–709.
[52] Xinming Shi, Leandro L. Minku, and Xin Yao. 2022. Adaptive memory-enhanced time delay reservoir and its memristive implementation. *IEEE Transactions on Computers* 71, 11 (2022), 2766–2777.

[53] Herbert Jaeger and Harald Haas. 2004. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science* 304, 5667 (2004), 78–80.
[54] Byung Joon Choi, Antonio C. Torrezan, John Paul Strachan, P. G. Kotula, A. J. Lohn, Matthew J. Marinella, Zhiyong Li, R. Stanley Williams, and J. Joshua Yang. 2016. High-speed and low-energy nitride memristors. *Advanced Functional Materials* 26, 29 (2016), 5290–5296.
[55] Chatchawit Aporntewan and Prabhas Chongstitvatana. 2001. A hardware implementation of the compact genetic algorithm. In *Proceedings of the 2001 Congress on Evolutionary Computation*, Vol. 1. IEEE, Los Alamitos, CA, 624–629.

Received 27 July 2022; revised 28 May 2023; accepted 9 May 2023