Surrogate-Assisted Cooperative Co-evolutionary Reservoir Architecture Search for Liquid State Machines

Yan Zhou, Yaochu Jin, *Fellow, IEEE*, Yao Sun, and Jinliang Ding, *Senior Member, IEEE*

***Abstract*—Liquid state machines (LSMs) are biologically more plausible than feedforward spiking neural networks for brain-inspired computing and neuromorphic engineering. However, optimizing and training complex recurrent network architectures in the reservoir of LSMs remains challenging. Most existing algorithms aim to adjust the synaptic strength only, without fundamentally modifying the reservoir architecture of LSMs. Recently, it has become popular to simultaneously optimize the architecture and parameters of the reservoir in LSMs. However, most existing architecture representation schemes are too restricted to discover more powerful architectures of LSMs. To address the above issue, this paper proposes a generative liquid state machine, whose reservoir architecture is evolved using a cooperative co-evolutionary algorithm whose weights are tuned by synaptic plasticity rules. To reduce the computation time for evolving the reservoir, random forest is adopted to assist the cooperative co-evolutionary algorithm, together with a data parallelism strategy. The proposed algorithm is assessed on three sequence classification benchmarks and our experimental results show that the proposed algorithm outperforms the state-of-the-art on the benchmark problems. Meanwhile, our analysis shows that the data parallelism strategy is effective in speeding up the evaluation process.**

***Index Terms*—Cooperative co-evolutionary algorithm, liquid state machines, neural architecture search, plasticity, random forest, surrogate-assisted optimization.**

Manuscript received 29 July 2022; revised 28 October 2022; accepted 18 November 2022. The work was Yaochu Jin was supported by Alexander von Humboldt Professor for AI endowed through the German Federal Ministry of Education and Research. This work was supported in part by the National key R&D plan of China, under Grant 2022YFB3304700, in part by the National Natural Science Foundation of China, under Grants 61988101, 62161160338, and 61733005, and in part by the Central Government Guides Local Science and Technology Development Funds under Grant 2022020522-JH6/1001. *(Corresponding author: Yaochu Jin.)*

Yan Zhou and Jinliang Ding are with the State Key Laboratory of Synthetical Automation for Process Industry, Northeastern University, Shenyang 110819, China (e-mail: yanzhou.ac@outlook.com; jlding@mail.neu.edu.cn).

Yao Sun is with the Shenyang Institute of Automation, Chinese Academy of Sciences, Shenyang 110169, China (e-mail: sunyao@sia.cn).

Yaochu Jin is with the State Key Laboratory of Synthetical Automation for Process Industry, Northeastern University, Shenyang 110819, China, also with the Faculty of Technology, Bielefeld University, 33619 Bielefeld, Germany, and also with the Department of Computer Science, University of Surrey, Guildford GU2 7XH, U.K. (e-mail: yaochu.jin@uni-bielefeld.de).

This article has supplementary downloadable material available at https://doi.org/10.1109/TETCI.2022.3228538, provided by the authors.

Digital Object Identifier 10.1109/TETCI.2022.3228538

I. Introduction

In RECENT years, neuromorphic computing has received increased attention due to its biologically plausible mechanisms and energy efficient hardware implementations [1], [2], [3]. Brain-inspired computing is a computing system imitating the brain functions, which is primarily applied in neuromorphic computing [4], [5]. Two primary brain-inspired models, namely spiking neural networks (SNNs) [6] and liquid state machines (LSMs) [7], have been widely investigated in neuromorphic computing and many machine learning algorithms have been proposed [1], [3]. By connecting a large number of spiking neurons with a complex structure, called reservoir, LSMs can store temporal information of the input in the neural dynamics and the recurrently connected synapses. Therefore, LSMs have been shown more powerful in processing sequence data than multi-layer SNNs [8]. Although LSMs with a randomly connected reservoir without further training can already achieve satisfactory learning performance, increasing evidence has been reported showing that optimization and training of the reservoir can further enhance the performance [9], [10], [11]. However, most existing learning algorithms are developed for feed-forward SNNs [8], and training the weights in the reservoir of the LSMs remains challenging due to the reservoir's recurrently connected complex architecture.

The primary challenge in training SNNs lies in the fact that the widely used gradient-based learning algorithms for analog artificial neural networks are not directly applicable to SNNs due to their discrete nature of the neural dynamics. Consequently, several supervised learning algorithms have been developed for SNNs concentrating on the classification of temporal patterns [8], [12]. For example, a threshold-driven plasticity is developed to deal with both rate and temporal coding spike trains efficiently [13]. Also, to extend these algorithms to LSMs, a recent work proposes to approximate the LSM with a recurrent SNN so that error back-propagation through time can be applied [11]. Besides, bio-inspired unsupervised learning methods can also be adopted to optimize the local neural response of the reservoir in LSMs. For instance, the Hebbian learning rule [14] is used to train hardware-based LSMs [15], and the spike-timing-dependent plasticity (STDP) learning rule [16] is adopted to train the deep-LSM for image classification [17] and video activity recognition [18]. However, the network parameters trained by global neural plasticity rules (unsupervised learning rules, see

Section IV-E) are influenced by the interference between different patterns [9]. Furthermore, such approaches can only optimize synaptic strength (weights of the neural network) without modifying the reservoir architecture, consequently restricting the performance enhancement of LSMs.

It has been widely recognized that the architecture of neural networks is critical to the cognitive competence in both biological brains and artificial systems [19], [20]. A large body of research on neural architecture search (NAS) has been carried out in the field of deep learning [20]. The deep neural networks are mainly encoded by block-based or cell-based architecture representation and various optimization strategies have been employed in NAS, such as reinforcement learning, evolutionary algorithms, and gradient-based approaches [20], [21]. One primary concern in performing NAS for large machine learning models on a large dataset is the prohibitive computational cost, restricting the performance of the NAS algorithms when limited computational resources are available. A recent survey provides an overview of various approaches to the reduction of computation time, e.g., employing the block-based search space, adopting efficient optimization algorithms, and reducing the computational burden for performance estimation [20].

Likewise, it has also been highlighted that NAS plays a critical role in improving the performance of LSMs, both in neuromorphic design [2] and reservoir computing [22]. Some preliminary studies have achieved better performance by designing LSM architectures based on NAS [10], [23], although many grand challenges remain to be solved. For instance, existing architecture representation schemes [10], [23] are highly restricted by the fixed local connections, the non-extendable network structure, and a limited search space. All these weaknesses prevent them from effectively improving the learning performance of LSMs. Additionally, the reservoir architecture of LSMs in existing architecture search methods is based exclusively on a random initialization, making them inefficient for exploring modular reservoir architectures. Therefore, inspired by the techniques for NAS of deep neural networks, developing efficient methods for automatically designing LSMs with a large-scale and modular architecture is essential yet challenging.

To meet the challenges discussed above, this study proposes several ideas. First of all, the block-based architecture representation for NAS can primarily be employed in the search for modular architectures of LSMs. While different convolutional and pooling blocks are widely used in deep neural networks, several typical topologies in complex networks have been frequently used for LSMs [24], including scale-free networks and small-world networks. These complex network models have been shown to be more effective than random networks for reservoir computing [25], [26]. Therefore, they may also be used as building blocks in searching large-scale modular architectures for LSMs [27]. However, the block-based representation of large-scale architectures may lead to a very high-dimensional search space, making an optimization algorithm for NAS less effective.

A common way to address the above challenge in NAS is to employ hierarchical architecture representations [28]. A hierarchical architecture representation [28] can further organize the blocks into sub-reservoirs, in which the blocks can share the same parameters, thereby reducing the search space. In addition, the intrinsic plasticity [29] and STDP [16], [30] allow LSMs to effectively adjust the dynamic threshold and synaptic strength according to the input stimulus. Because of this, these parameters need not be encoded in the representation, further reducing the search space. Consequently, this study proposes a block-based hierarchical architecture representation scheme for searching large-scale modular architectures of LSMs with the help of neural plasticity rules. We term the proposed method generative Liquid State Machine, GLSM for short.

Although the proposed hierarchical architecture representation reduces the dimensional of the search space, the NAS for GLSM remains a high-dimensional optimization problem. Many existing studies have employed a divide-and-conquer evolutionary algorithm, namely cooperative co-evolutionary algorithm (CCEA) [31] to successfully solve high-dimensional architecture optimization problems of artificial neural networks [32], [33]. Meanwhile, the decision variables can be divided into several groups based on the hierarchical architecture representation. Therefore, we adopt the CCEA to decompose the architecture optimization of the GLSM into several low-dimensional subproblems under the framework of evolutionary NAS (ENAS) [21].

To reduce the computational cost for performance evaluations of the neural architectures in ENAS, this work employs the random forest as the surrogate model for performance estimation [34], [35] in the CCEA based architecture search. Consequently, a surrogate-assisted cooperative co-evolutionary (SACC) algorithm is proposed as the optimization strategy of the ENAS framework to accelerate the search of large-scale modular structures of the GLSM. Thus, the overall framework proposed in this work is called SACC-GLSM.

Note that, for the performance evaluation of the GLSM, each data sampled in the classification dataset needs to be converted into a spike train with a series of binary numbers. Since a real number requires many binary numbers to represent, computing with spike trains consumes more computing time on X86-platform CPUs than computing with data composed of real numbers. Therefore, it will take additional effort to extract the states of each spike train from the spiking neurons and the plasticity-driven training of the reservoir. To overcome this bottleneck, this work employs the data parallelism [36] through the distributed parallel framework suggested in [37] to further speed up the training of the GLSMs.

The performance of SACC-GLSM is tested on three sequence classification problems, namely the KTH action recognition benchmark [38], the HAR action recognition benchmark [39], and the dynamic vision sensor (DVS) [40] based neuromorphic benchmark (N-MINIST) [41]. The experimental results indicate that the proposed method achieves state-of-the-art performance with less time consumption. Regression analysis of two effective graph measures corresponding to the evaluated performance, provided in the supplementary material, indicates that a long average path length and a low clustering coefficient will lead to better performance. The major contributions of this work are summarized as follows.

- A block-based hierarchical architecture representation scheme is proposed for generating large-scale modular

LSMs. A variety of typical complex network models are encoded as building blocks to improve the performance of the GLSMs. Neural plasticity rules are adopted in GLSMs to reduce the search space of the block-based representation.
- A random forest assisted CCEA is developed for efficient search of large-scale modular GLSMs. The CCEA decomposes the high-dimensional search space, while the random forest serves as a computationally efficient surrogate to assist the CCEA based search of GLSMs to reduce the computation time.
- Finally, a data parallelism strategy for further speeding up the performance evaluation process of GLSM is adopted. To the best of our knowledge, this is the first time that data parallelism is employed in reservoir architecture search of LSMs.

The remainder of this paper is organized as follows. Section II presents the related work, which is followed by Section III that briefly introduces the overall framework of SACC-GLSM. Section IV elaborates the proposed GLSM and Section V describes the details of the SACC algorithm. The experimental settings are presented in Section VI, together with the experimental results and discussions. Finally, Section VII concludes this paper.

II. Related Work

This section introduces the related research on reservoir architectures and NAS. Although these methods are closely related to the present work, none of them is ready for the search of large-scale modular reservoirs for LSMs. Little existing work on optimization of the reservoir architecture has taken into account the particular demands of the LSMs in terms of modularity, plasticity, and spike-based information processing. Therefore, this work aims to fill the gap by developing an efficient NAS algorithm that can generate high-performance LSMs with a modular reservoir architecture whose weights are tuned by neural plasticity rules.

A. Reservoir Architecture

Some studies have been reported the influence of the architecture and parameters of the reservoir on the performance of LSMs. A large body of early research on LSMs has been dedicated to investigating the performance of LSMs when one of the popular complex network models is adopted for the reservoir. For example, the complex network models with irregular and modular structures can improve the performance of LSMs on discrimination tasks [26]. Most recently, the relationships between the graph measures of complex network models and the predictive performance has been analyzed [42].

Likewise, many studies of the reservoir architecture have been carried out for echo state networks (ESNs) that use continuous neurons. For example, the optimal modular structure in the reservoir is proved to be better for memory tasks [43]. The DeepESN designs the depth of ESN layers by proposing an automatic algorithm based on spectral analysis of the reservoir [44]. Additionally, complex network models are also employed in ESNs. The performance of approximating highly complex dynamic systems is considerably improved by a highly clustered echo state network which naturally grows with scale-free and small-world rules [45].

B. NAS

NAS methods have been extended to the search for optimal reservoir architectures of LSMs. For instance, a block-based search space is designed for searching the optimal reservoir architectures of LSMs [23]. However, it only supports a fixed number of neurons, and all blocks share the same parameters and are updated synchronously. Another study [10] proposes a surrogate-assisted evolutionary search strategy in order to reduce the computation cost in performance estimation. However, the architecture representation scheme of [10] is inflexible and cannot facilitate the generation of modular reservoirs.

Based on ENAS, the CCEA is employed to evolve the modular architecture of the GLSM. Many studies show that the CCEA is effective in optimizing neural networks with a high-dimensional search space. For instance, the evolutionary deep neural networks optimized by the CCEA reduce the execution time by separating co-evolutionary strategies into two different populations [32]. Meanwhile, COVNET is proposed to co-evolve subnetworks with minimal constraints rather than evolving entire networks, hence achieving better generalization with smaller networks [33].

Finally, different methods have been investigated to effectively reduce the computational cost for NAS. For example in ENAS [21], a random forest surrogate [34] is utilized as the end-to-end performance predictor to accelerate the performance estimation in the off-line scenario. Another performance predictor proposed in ReNAS [46] uses the adjacency matrix of neural architectures and computational power features to predict the pairwise ranking of candidate architectures. A recent method, termed RelativeNAS, benefits from both the advantages of ENAS and gradient-based NAS to achieve state-of-the-art performance in convolutional neural networks (CNNs) [47]. In [48], each mini-batch randomly samples an individual in the parent population, significantly reducing the computational cost of ENAS.

III. Framework Overview

This section provides an overview of the proposed framework of SACC-GLSM. SACC-GLSM is mainly composed of the architecture representation scheme, which defines the search space $\mathcal{S}$, the optimization strategy $\mathcal{O}$, and the performance estimation strategy including both the original function $\mathcal{F}$ and the performance predictor $\mathcal{RF}$. The original function $\mathcal{F}$ means the real evaluation (expensive training) of the GLSM, and the performance predictor $\mathcal{RF}$ is the random forest surrogate. Finding the optimal architecture of the GLSMs can be considered as the optimization problem defined below in (1),

$$\arg\max_{x} \mathcal{F}(x, \mathcal{D}) \qquad \text{s.t.} \quad x \in \mathcal{S} \tag{1}$$

where x denotes the candidate solution selected by the optimization algorithm $\mathcal{O}$, and $\mathcal{D}$ denotes the classification dataset.

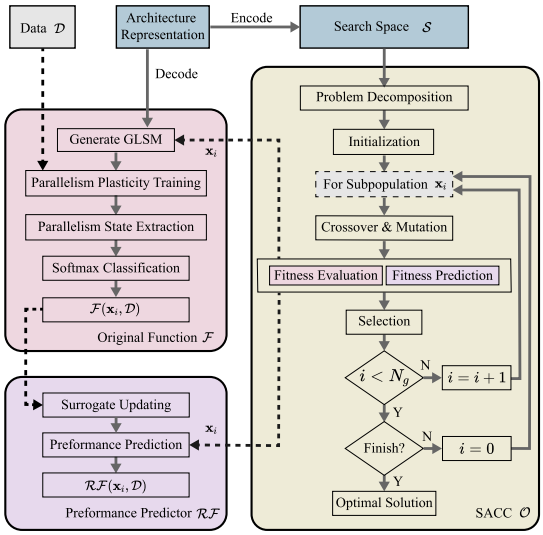


Fig. 1. An illustration of the framework of SACC-GLSM. The right side is the optimization process of SACC-GLSM, and the performance estimation strategy of SACC-GLSM is shown on the left.

The optimization process of SACC-GLSM is shown on the right side of Fig. 1. The search space $\mathcal{S}$ is encoded by the block-based hierarchical architecture representation scheme. According to the search space $\mathcal{S}$, the SACC algorithm $\mathcal{O}$ decomposes the original problem into N_g subproblems and initializes the subpopulations and the surrogate. For each subpopulation i, the individuals $\mathbf{x}_i$ are evolved by crossover and mutation. Note that, in the SACC algorithm, the subpopulations refer to the populations of a subproblem. The fitness of the individuals $\mathbf{x}_i$ is evaluated by the original function $\mathcal{F}$ or predicted by performance predictor $\mathcal{RF}$. The best individuals are selected as the parents of the next generation. After all subpopulations are evolved, if the termination condition is satisfied, the best solution is the optimal architecture of the GLSM.

The left side of Fig. 1 illustrates the performance estimation strategy of SACC-GLSM. With respect to fitness evaluations, the GLSM is generated according to the parameters decoded from the individuals $\mathbf{x}_i$. The classification accuracies $\mathcal{F}(\mathbf{x}_i, \mathcal{D})$ are evaluated via parallelism plasticity training, parallelism state extraction, and softmax classification. Meanwhile, the evaluated classification accuracies $\mathcal{F}(\mathbf{x}_i, \mathcal{D})$ are used to update the surrogate model. The surrogate, also termed performance predictor $\mathcal{RF}$, predicts the approximate accuracies $\mathcal{RF}(\mathbf{x}_i, \mathcal{D})$ of the individuals $\mathbf{x}_i$. Then the accuracies $\mathcal{F}(\mathbf{x}_i, \mathcal{D})$ and $\mathcal{RF}(\mathbf{x}_i, \mathcal{D})$ are both used as the fitness to select candidate solutions of the next generation in the SACC algorithm $\mathcal{O}$.

IV. Generative Liquid State Machine (GLSM)

This section starts with introducing the GLSM architecture before delving into the details of the block-based hierarchical architecture representation. Then, the neural dynamic models of the GLSM and the adopted plasticity rules are introduced to demonstrate how spike trains are integrated with spiking neurons. Finally, an illustration of the data parallelism plasticity training and state extraction processes is given.

A. Architecture of the GLSM

Generally, LSMs are composed of encoding layer, reservoir layer and readout layer. The encoding layer is responsible for converting the input data into spike trains. The reservoir layer contains complex network structures with recurrent connections. The reservoir is a critical component for converting the input spike trains into the reservoir states. The readout layer extracts the states of the reservoir as high-dimensional features in order to predict the classification accuracy.

To promote the efficacy and facilitate the search efficiency, the proposed GLSM divides the reservoir into several blocks and organizes the blocks as hierarchical sub-reservoirs. A diagram of the architecture of the GLSM is given in Fig. 2. Instead of a single spiking neuron, the GLSM regards the block composed of spiking neurons and inner connected synapses as the fundamental computing component. The spiking neurons in each block are connected based on four complex network models, namely random network model, scale-free network model, small-world network model, and a three-layer network model. Each block contains two categories of spiking neurons, i.e., excitatory neurons and inhibitory neurons. The excitatory neurons create positive post-synaptic potentials (PSPs), whereas the inhibitory neurons produce negative PSPs. The number of excitatory and inhibitory neurons follows the ratio of $4:1$ [7]. Every four blocks are organized as a sub-reservoir, and the reservoir consists of four sub-reservoirs. There are four block types and four sub-reservoir types, as shown in Fig. 3. The blocks are connected through a group of synapses, which is referred to as a pathway. The block chooses several neurons as the output and input neurons based on topological sorting. The input and output blocks of the reservoir layer are determined by the topological sorting of the four sub-reservoirs. The pathway creates the synapses between the output neurons and the input neurons of two blocks with a probability of p_{res}.

The bottom of Fig. 2 demonstrates the computational process of the GLSM. The encoding layer is composed of a single block that converts sequence data to spike trains by encoding neurons. The number of neurons in the encoding block depends on the number of input features. The pathways between the encoding block and the blocks of the reservoir layer are connected with the connection probability p_{en}. Finally, the readout layer is composed of a single block with readout neurons. Each readout neuron converts the neural dynamic state of one spiking neuron in the reservoir into a real number. The number of readout neurons is the same as the total number of spiking neurons in the reservoir layer. The synapses of the pathways between the blocks of the reservoir layer and readout block are one-to-one connections. In other words, each spiking neuron in the reservoir layer connects a corresponding readout neuron in the readout layer.

B. Block-Based Architecture Representation

Block-based architecture representation is an effective indirect encoding scheme for NAS [21], [49]. Each block holding an unique network structure is responsible for fundamental information processing in the reservoir. The network structure

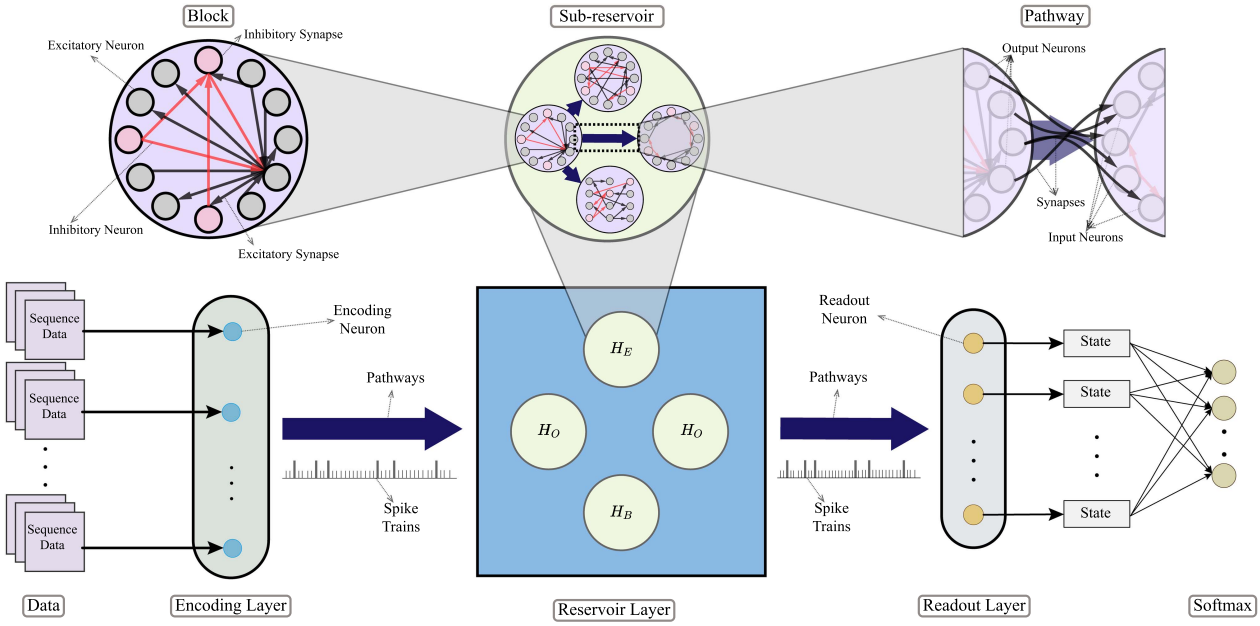


Fig. 2. A diagram of the GLSM architecture. At the bottom of the diagram is the illustration of different layers of the GLSM. The encoding layer converts the input data into the spike trains, and the spike trains are subsequently injected into the reservoir layer. The reservoir states are extracted by the readout layer and then utilized by the softmax to calculate classification accuracy. The top of the diagram demonstrates the three major components of the reservoir layer, which are the block, the sub-reservoir, and the pathway, respectively. The reservoir contains four sub-reservoirs and each sub-reservoir is composed of four blocks. The blocks contain several spiking neurons connected by various types of network structures. The pathway connects the output neurons and the input neurons of two blocks with a group of the synapses. The neurons in the block are split into excitatory neurons and inhibitory neurons.

of the block can be represented by a directed graph $G(V, E)$, where vertices $V = \{v_i, i \in 1, \ldots, N\}$ denote N spiking neurons and edges $E = \{e_{i,j}, i \in 1, \ldots, N, j \in 1, \ldots, N\}$ denote synapses from the spiking neuron v_i to v_j. The network structure can be mathematically represented by a adjacent matrix $A = \{a_{i,j}, i \in 1, \ldots, N, j \in 1, \ldots, N\}$, where $a_{i,j} = 1$ denotes existing synapse $e_{i,j}$, and $a_{i,j} = 0$ denotes no synapse $e_{i,j}$.

The network structure of the block is highly constrained if only the random blocks are used. This work employs various complex network models to explore more diverse and effective modular structures. Due to the limited computational resources, we employ four typical complex network models to generate the network structure of the block. The network structures of the four complex network models have different properties, which we believe sufficient for this study to validate the impact of network structure on the classification performance of the GLSM. The four block types are random blocks, scale-free blocks, small-world blocks, and three-layer blocks. The examples of the four block types are shown on the left of Fig. 3(a), with twelve neurons in each block. The network structures of different blocks show different characteristics. The synapses of the random block are uniformly distributed. The scale-free block contains a major hub with a larger degree. The neurons of the small-world block connect only with neighbours, and the three-layer block has no synapses between the neurons in the same layer. In the following, we introduce the details of the four block types. The input and output neurons of the block are determined by topological sorting and strongly connected components.

1) Random Block: The original LSM primarily employs the random network as the network structure of the reservoir. The synapse $e_{i,j}$ is generated with a connection probability of p_{rd} following a uniform distribution. If the network structure of the block is the random network, the synapse $a_{i,j} = 1$ will be uniformly distributed in the adjacent matrix A.

2) Scale-Free Block: Scale-free networks are the most famous network model existing in the internet network and many real-world networks. Some hubs in the scale-free network have more connections than the rest of the vertices in the network structure. The degree distribution $p(d)$ of the scale-free network asymptotically follows the power-law $p(d) \sim k^{-\gamma}$, where d is the degree of the vertex and γ is a parameter generally with the range $2 \leq \gamma \leq 3$. The Barabási-Albert model is the most widely known generative model of the scale-free network for an undirected graph with $\gamma \approx 3$ [50]. Since the network structure of the block is a directed graph, this work employs a directed scale-free network [51] model to generate the scale-free block. Algorithm 1 illustrates the scale-free block generation algorithm based on three connection probabilities p_α, p_β and p_γ, which are critical parameters for generating the scale-free block.

3) Small-World Block: Another network model defined by the distance between vertices is the small-world network, in which all vertices are more likely connected to their neighbors. The small-world network has a small average path length and a high clustering coefficient, and the average path length between any two vertices is proportional to the logarithm of the number of vertices. The small-world network is found in many real-world networks, especially in the network structure of a biological brain [50]. Therefore, this work adopts a direct small-world network model as one block network structure. The detail of generating the proposed small-world block is described

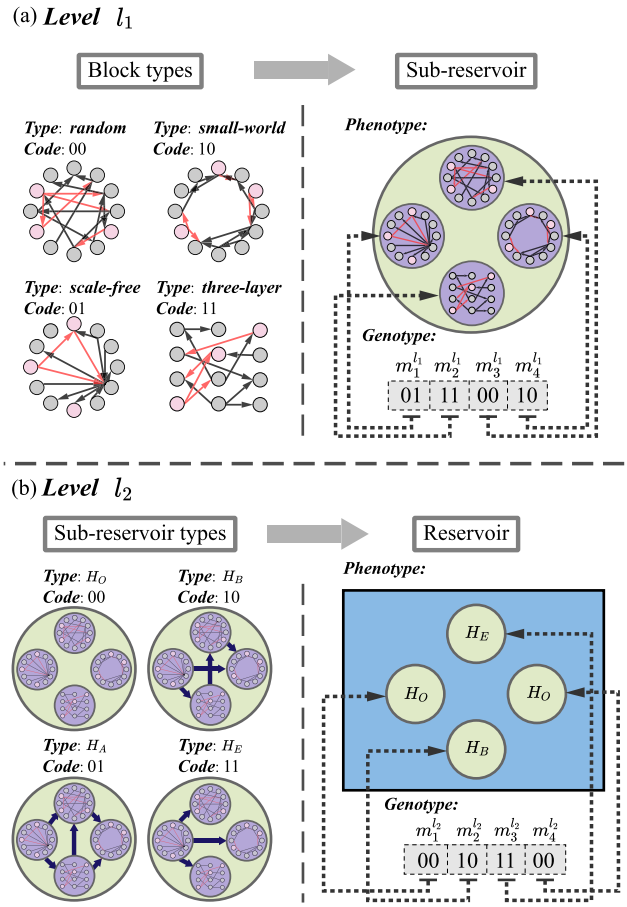


Fig. 3. An illustration of the l_1-th level and the l_2-th level of the hierarchical architecture representation. The l_1-th level contains four block types and the l_2-th level includes four sub-reservoir types. Each level shows an example of mapping the genotype to the phenotype.

in Algorithm 2, where p_f and p_b represents the connection probability of v_i to v_j and v_j to v_i, respectively. The proposed small-world network model connects two vertices according to the connection probability and their distance. As a result, a pair of vertices will be more likely connected if their distance is short. Additionally, no connections can be established when the distance between two vertices exceeds the distance threshold θ.

4) Three-Layer Block: Finally, the three-layer network, which is the most common network structure for conventional neural networks, is also adopted in the block as a candidate network model. The neurons of three-layer blocks are divided evenly into three layers. Each neuron can only connect to the neurons in other layers with the input connection probability p_{in} and output connection probability p_{out}. The connection probabilities p_{in} and p_{out} decrease with the increase of the in-degree d_{in} and out-degree d_{out} of the neurons. For each two neurons v_i and v_j in the different layers, the synapse $e_{i,j}$ is connected following the connection probability $p_{out} \times p_{in} \times \sigma^{(d_{out}(v_i)+d_{in}(v_j))}$, where σ is the probability decay coefficient.

5) Determination of the Input and Output of a Block: Once the network structure is confirmed, the input and output neurons that communicate with other blocks must be specified. To ensure that the input spikes flow through as many neurons as possible, this work uses topological sorting to determine the sequence of the neurons. The vertices at the start and the end of the sorted topological sorting sequence are identified as the input and output neurons, respectively. However, the topology of the network is a cyclic graph, which should guarantee to be directed acyclic graph (DAG) for topological sorting. The cycle in the graph can be aggregated into strongly connected components [52] as a single vertex cluster V_c to transfer the graph to be acyclic graph $G(V, V_c, E)$. If the V_c are at the start

Algorithm 1: Scale-free Block Generation.

Input: The number of vertices N, the connection probability p_α, p_β and p_γ.
Output: The adjacent matrix A.

Initialize $A = \{0\}_{N\times N}$;
Add two initial vertices v_0, v_1;
The current number of nodes $n = 2$;
Add an edge $e_{0,1}$, set $a_{0,1} = 1$;
while $n < N$ **do**
  Calculate the in-degree d_{in} and out-degree d_{out};
  # Add connections with probabilities p_α, p_β and p_γ. #
  if *With the probability* $\frac{p_\alpha}{p_\alpha+p_\beta+p_\gamma}$ **then**
    Select an input vertex v_j with the probability $\frac{d_{in}(v_j)}{m+n}$;
    Add new vertex v_{n+1} and the edge $e_{n+1,j}$;
    Set $a_{n+1,j} = 1$, $n = n + 1$;
  else if *With the probability* $\frac{p_\beta}{p_\alpha+p_\beta+p_\gamma}$ **then**
    Select an input vertex v_j with the probability $\frac{d_{in}(v_j)}{m+n}$;
    Select an output vertex v_i with the probability $\frac{d_{out}(v_i)}{m+n}$;
    if $e_{i,j}$ *is not exist* **then**
      Add new edge $e_{i,j}$;
      Set $a_{i,j} = 1$;
    end
  else if *With the probability* $\frac{p_\gamma}{p_\alpha+p_\beta+p_\gamma}$ **then**
    Select an output vertex v_i with the probability $\frac{d_{out}(v_i)}{m+n}$;
    Add new new vertex v_{n+1} and edge $e_{i,n+1}$;
    Set $a_{i,n+1} = 1$, $n = n + 1$;
  end
end
return A;

Algorithm 2: Small-world Block Generation.

Input: The number of vertices N, the connection probability p_f and p_b, the distance threshold θ.
Output: The adjacent matrix A.

Initialize $A = \{0\}_{N\times N}$;
Add N initial vertices;
# Add connections with probabilities p_f, p_b and the distance threshold θ. #
for *Each two vertices* v_i *and* v_j **do**
  $decay = \begin{cases} 0 & |i-j| \geq N \times \theta \\ 1 - \frac{|i-j|}{N\times\theta} & |i-j| \leq N \times \theta \end{cases}$
  if *With the probability* $p_f \times decay$ **then**
    Add an edge $e_{i,j}$, set $a_{i,j} = 1$;
  end
  if *With the probability* $p_b \times decay$ **then**
    Add an edge $e_{j,i}$, set $a_{j,i} = 1$;
  end
end
return A;

or end of the sequence, one vertex of the V_c is chosen as the input or output neuron. In addition, when the total input and output neurons are less than 30% of all the neurons, more neurons are randomly chosen until the number of the input and output neurons reaches 30%. To illustrate the proposed method, an example of determining the input and output neurons is shown in Fig. S1 in the Supplementary material, which is composed of twelve neurons in the block.

C. Hierarchical Architecture Representation

This work employs the hierarchical architecture representation scheme [28] to further organize the blocks into the reservoir and reduce the computational burden for the block-based ENAS. The hierarchical architecture holds L levels $\{l_i, i = 1, \ldots, L\}$ and each level l_i consists of M basic components $\{m_j^{l_i}, j = 1, \ldots, M\}$, also denoted as motifs. The hierarchical architecture representation scheme constructs the higher level motif using the motifs in the lower level. In other words, the motif of a higher level contains M motifs of the lower level, which can be represented as $m_j^{l_{i+1}} \leftarrow \{m_j^{l_i}, j = 1, \ldots, M\}$. Each level has several motif types that are used as the coordinates of the motif structure. The type of motif $m_j^{l_i}$ is selected from the coordinate motif types of the level l_i according to the genotype.

As shown in Fig. 3, this work only employs $L = 2$ levels in the reservoir layer and each level has $M = 4$ motifs since our computational resource is limited. In level l_1, motif $m_j^{l_1}$ represents a block whose network structure is selected from the four block types described in the Section IV-B. After all motif types in level l_1 are confirmed, motifs $\{m_j^{l_1}, j = 1, \ldots, 4\}$ are aggregated into motif $m_j^{l_2}$. Motif $m_j^{l_2}$, also called sub-reservoir, connects to $\{m_j^{l_1}, j = 1, \ldots, 4\}$ according to one of the pre-defined sub-reservoir types. In level l_2, four DAG graphs are regarded as the coordinate sub-reservoir types, as shown on the right side of Fig. 3(b), which are denoted by H_O, H_A, H_B, and H_E, respectively. Notably, the four sub-reservoir types are chosen from the typical DAG graphs when $M = 4$. If level l_2 contains more than $M = 4$ motifs, the coordinate sub-reservoir types also increase. However, considering the computational burden, $M = 4$ is selected to illustrate the effectiveness of the proposed hierarchical architecture representation scheme. Finally, all sub-reservoirs are aggregated into a whole reservoir without inner pathways. The input of the sub-reservoir is the head block of the DAG graph according to the topological sorting. The pathways between the encoding layer and the reservoir layer actually connect the encoding block to the input blocks of the sub-reservoirs.

Fig. 3 also shows an example of mapping the genotype to the phenotype based on the hierarchical architecture representation. Motifs $\{m_j^{l_1}, j = 1, \ldots, 4\}$ are generated according to the genotype $|01|11|00|10|$, which means the phenotype of the sub-reservoir consists of the scale-free block, the three-layer block, the random block, and the small-world block. In level l_2, each motif $m_j^{l_2}$ contains the same motifs $\{m_j^{l_1}, j = 1, \ldots, 4\}$. Meanwhile, according to the genotype $|00|10|11|00|$, the motifs of the sub-reservoirs $\{m_j^{l_2}, j = 1, \ldots, 4\}$ are connected based on the type of H_O, H_B, H_O and H_E, respectively.

D. Neural Dynamic Models

The spiking neurons are the basic computation unit of each block in the GLSM. Different from the neurons of conventional neural networks, the data processed by spiking neurons are a sequence of pulses with various time intervals, called spike train $S_i(t)$, which is mathematically expressed as follows:

$$S_i(t) = \sum_f \delta\left(t - t_i^{(f)}\right), \tag{2}$$

where $\delta\left(t - t_i^{(f)}\right)$ is the Dirac function describing a single pulse and $t_i^{(f)}$ is the fire time of the pre-synaptic neuron i.

The most common neural dynamic model of the spiking neuron is leaky integrate-and-fire (LIF) model [53].

$$\tau_n \frac{du_i(t)}{dt} = -u_i(t) + u_{rest} + \sum_j \kappa_{ij}(t), \tag{3}$$

where τ_n is the membrane time constant. The post-synaptic neuron fires a spike when u_i reaches the threshold $T_i(t)$, and then the membrane potential u_i is reset to rest membrane potential u_{rest}. The post-synaptic neuron keeps silent in the refractory period t_{ref} after firing. The PSPs caused by pre-synaptic neuron j is described as the synaptic dynamic model $\kappa_{ij}(t)$.

$$\kappa_{ij}(t) = \sum_f w_{ij} \cdot e^{(-\frac{t}{\tau_s})} \cdot \delta\left(t - t_j^{(f)}\right) + I_0, \tag{4}$$

where τ_s is the decay time constant, w_{ij} refers to the synaptic strength of synapse from pre-synaptic neuron j and I_0 is inherent injection to the post-synaptic neuron.

E. Plasticity Rules

Plasticity is the most important learning mechanism found in the biological brain, contributing to the construction of knowledge and memory [19], [29]. The learning process starts after the initialization of the neural network architecture. The plasticity rules adjust the network parameters according to the input stimuli. In this study, the intrinsic plasticity rule [11], [29] and the synaptic plasticity rules [16], [30] are adopted to adjust the neural threshold $T_i(t)$ and the synaptic strength $w_{i,j}$, respectively.

1) Intrinsic Plasticity Rule: Intrinsic plasticity is a homoeostatic learning rule for spiking neurons to adjust the firing state. According to the input spike trains, the intrinsic plasticity rule adjusts the neural threshold $T_i(t)$ following (5),

$$\tau_{th} \frac{dT_i(t)}{dt} = -T_i(t) + T_0 + \Delta T \cdot \delta\left(t - t_i^{(f)}\right), \tag{5}$$

where the τ_{th} is the time constant of the dynamic threshold, T_0 is the basic threshold, ΔT denotes the threshold increment when the neuron v_i fires.

The crucial contribution of the intrinsic plasticity rule is to prevent the saturation of the firing state. The more neurons fire, the more difficult for a post-synaptic neuron to fire the next spike in the decay time τ_{th}. Thus, it maps the input spikes and the output spikes with a non-linear relationship. Similar to the activation function of conventional neural networks, the intrinsic

plasticity rule can be regarded as the spiking activation function of spiking neurons.

2) Synaptic Plasticity Rules: Synaptic plasticity plays an important role in learning and cognition in neural networks. Synaptic plasticity rules change the synaptic strength to enhance or inhibit the stimuli injected into the post-synaptic neuron. The most widely used synaptic plasticity rule for SNNs is spike-timing-dependent plasticity (STDP) [16]. The STDP rule modifies the synaptic strength depending on the temporal correlation between pre-synaptic and post-synaptic spikes. The synaptic strength increases when the pre-synaptic neuron fires earlier than the post-synaptic neuron; otherwise the synaptic strength decreases. Similar to STDP, a more stable plasticity rule, denoted the multiplicative STDP rule, considers not only the temporal correlation but also the upper and lower bounds of synaptic strength [30]. Both the STDP rule and the multiplicative STDP rule represented in (6) are adopted in the GLSM.

$$\Delta w_{ij} = \begin{cases} A_P \cdot (w_{i,j}^{\max} - w_{i,j})^{\gamma} \cdot exp\frac{-\Delta t}{\tau_p} \\ -A_D \cdot (w_{i,j} - w_{i,j}^{\min})^{\gamma} \cdot exp\frac{-\Delta t}{\tau_p} \end{cases}, \tag{6}$$

where $\Delta w_{i,j}$ is the strength increment of $w_{i,j}$, $w_{i,j}^{\max}$ and $w_{i,j}^{\min}$ are the upper and lower bound of $w_{i,j}$, respectively. $A_P = 0.01$ and $A_D = 0.0105$ are the incremental coefficient, τ_p is the time constant of the plasticity rule, and γ is the plasticity type parameter. In particular, when $\gamma = 1$, (6) represents the STDP rule, while when $\gamma = 0$, (6) represents the multiplicative STDP rule. To clearly illustrate the effect of the synaptic plasticity, examples of the synaptic strength distribution of the GLSM after being trained are shown in Fig. S2 in the Supplementary material. These examples show that the synaptic strength is distributed around $w_{i,j}^{\max}$ and $w_{i,j}^{\min}$ after trained by the STDP. By contrast, the synaptic strength is distributed around $(w_{i,j}^{\max} + w_{i,j}^{\min})/2$ after trained by the multiplicative STDP.

F. Parallelism State Extraction and Plasticity Training

The reservoir states reflect the counts and times of spikes emitted by the neurons in the reservoir layer, which represent the input spatio-temporal features of the data. The readout neurons extract the reservoir states through the LIF model without the firing threshold and reset potential [10]. Without firing spikes, the readout neurons integral the input spikes into the membrane potential u_{read} under the membrane time constant τ_{read}. The membrane potential continues to increase until the spike train ends, at which point the final membrane potential is considered to be the high-dimensional state of the reservoir. The reservoir state is exacted for every data of the dataset. After injecting all sequence data into the network, all reservoir states are fed into the softmax to evaluate the classification accuracy of the GLSM.

Conventional neural networks process the data in the form of real numbers, and a real number is represented by multiple binary numbers in the spike train. Although the GLSM is hardware friendly, computing with a real number by CPUs consumes almost the same time with a binary number. Therefore, simulating the GLSM on the CPU of the X86-platform is more time-consuming than conventional neural networks. Consequently, the time consumption bottleneck of evaluating the GLSM is the plasticity training and the state extraction process for the classification dataset.

The data parallelism can speed up the evaluation of the GLSM. To implement the data parallelism, this work applies a parallel distributed framework of Ray [37] in the plasticity training and state extraction processes using data parallelism. Each data in the dataset $\{d_i, i = 1, 2, \ldots, N_d\} \in \mathcal{D}$ is converted into a spike train by the encoding layer of GLSM. The data parallelism framework creates N_c processors to deal with the data in parallel. The number of the data for one processor is $\left\lceil \frac{N_d}{N_c} \right\rceil$. In the plasticity training process, each processor duplicates the initial GLSM for the data parallelism plasticity training. The trained synaptic strength of the reservoir is aggregated from the N_c processors defined in (7),

$$\mathcal{W} = \sum_{i}^{N_c} \left(\frac{W_i}{N_c} \right), \tag{7}$$

where $\mathcal{W}$ denotes the aggregated synaptic strength of the reservoir, and W_i denotes the trained synaptic strength of the duplicated reservoir in processor i. After the plasticity training process, the N_c processors duplicate the trained GLSM to exact the reservoir states in parallel. In general, the total time consumption is only approximately $\frac{1}{N_c}$ of the time consumption without the data parallelism.

V. Surrogate-Assisted Cooperative Co-Evolutionary Neural Architecture Search

This study aims to explore the possibilities of evolving the GLSM to achieve the optimal architecture for classification tasks. The architecture of the GLSM is optimized based on ENAS [21], which can be seen as an expensive black-box optimization problem. This section introduces the genotype settings of the GLSM and the proposed SACC algorithm.

A. Genotype Settings

The genotype of the GLSM represents the critical parameters of the block-based hierarchical architecture representation scheme. The GLSM is generated according to the parameters decoded from the genotype. In this study, the genotype contains 50 hybrid decision variables. The range, coding type and description of the decision variables are listed in Table I. Each decision variable is adaptively decoded by the corresponding decoding function according to the coding type listed in Table I. The range of the decision variables is decided by pilot studies and recommendations in [10]. The decision variables in Table I are shown in Fig. 4. Fig. 4(a) illustrates a diagram of the GLSM genotype settings. According to the architecture representation scheme of the GLSM, there is a strong correlation between the parameters of the same layer. Therefore, the genotype can be naturally decomposed into three or six groups. When the genotype is divided into three groups, the motifs of $\{m_i^{l_1}, i = 1, 2, 3, 4\}$ are regarded as one group. The three groups of decision variables shown in Table I and Fig. 4 are termed as "Encoding & Readout," "Reservoir" and "Blocks". Also, it should be noticed that the

TABLE I
THE PARAMETER SETTINGS IN THE GENOTYPE OF THE GLSM

	Parameter	Range	Type	Description
Encoding & Readout	γ_{en}	$[0, 1]$	Integer	The plasticity type of the synapses from the encoding layer to the reservoir layer.
	τ_{pen}	$[1ms, 100ms]$	Integer	The plasticity time constant of the synapses from the encoding layer to the reservoir layer.
	τ_{sread}	$[1ms, 10ms]$	Integer	The delay time constant of the synapses from the reservoir layer to the readout layer.
	α_{en}	$(0, 1]$	Real	The initial strength amplification of the synapses from the encoding layer to the reservoir.
	p_{en}	$[0.001, 0.2]$	Real	The connection probability of the synapses from the encoding layer to the reservoir layer.
Reservoir	mt_{res}	8 bits	Binary	The motif types of $\{m_i^{l_2}, i = 1, 2, 3, 4\}$.
	γ_{res}	$[0, 1]$	Integer	The plasticity type of the synapses between the blocks in the reservoir layer.
	τ_{pres}	$[1ms, 100ms]$	Integer	The delay time constant of the synapses between the blocks in the reservoir layer.
	α_{res}	$(0, 1]$	Real	The initial strength amplification of the synapses between the blocks in the reservoir layer.
	p_{res}	$(0, 0.9]$	Real	The connection probability of the synapses between the blocks in the reservoir layer.
Blocks	mt_{bl}	2 bits	Binary	The motif type $m_i^{l_1}$.
	γ_{bl}	$[0, 1]$	Real	The plasticity type of the synapses in the block $m_i^{l_1}$.
	N	$[15, 300]$	Integer	The number of the neurons in the block $m_i^{l_1}$.
	τ_{th}	$[10ms, 150ms]$	Integer	The threshold time constant of the neurons in the block $m_i^{l_1}$.
	τ_{sbl}	$[1ms, 10ms]$	Integer	The delay time constant of the synapses in the block $m_i^{l_1}$.
	τ_{pbl}	$[1ms, 100ms]$	Integer	The plasticity time constant of the synapses in the block $m_i^{l_1}$.
	α_{bl}	$(0, 1]$	Real	The initial strength amplification of the synapses in the block $m_i^{l_1}$.
	p_{bl}^a, p_{bl}^b, p_{bl}^c	$[0.01, 0.3]$ $[0.1, 1]$ $[0.3, 0.8]$ $[0.5, 1]$	Real	The synapse connection probability : Random block: p_{rd}, -, -. Scale-free block: $p_\alpha, p_\beta, p_\gamma$. Small-world block: p_f, p_b, θ. Three-layer block: p_{in}, p_{out}, σ.

* *Note: p_{bl}^a, p_{bl}^b and p_{bl}^c represent the connection probabilities of the complex network models in the genotype, the ranges of which are depending on the chosen network model of $m_i^{l_1}$.*

variables of $\{m_i^{l_1}, i = 1, 2, 3, 4\}$ are independent of each other. As shown in the "Block" part of Table I, all variables of $m_i^{l_1}$ are used to generate the block, where p_{bl}^a, p_{bl}^b and p_{bl}^c represent the connection probabilities of the selected complex network models. For example, if $m_i^{l_1}$ is a scale-free block, p_{bl}^a, p_{bl}^b and p_{bl}^c represent p_α, p_β and p_γ, respectively, with the range of $[0.1, 1]$. Specifically, the symbol "-" means when $m_i^{l_1}$ is a random block, only p_{bl}^a is used as p_{rd}.

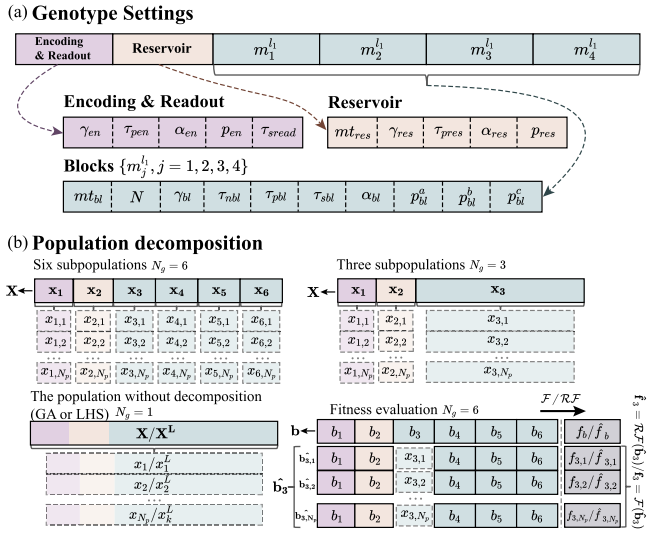


Fig. 4. A diagram of the genotype settings and the population decomposition for the ENAS-based GLSM.

B. CCEA

The CCEA decomposes the original problem into several low-dimensional subproblems [54]. Each subproblem is independently optimized with a subpopulation to obtain partial solutions. The individual of the subpopulation is evaluated in cooperation with the other subpopulations. The best partial solutions are combined to be a complete solution of the original problem. The fitness of the complete solution is then utilized to determine the fitness of the corresponding partial solutions [31], [55].

Accordingly, the essential aspects of the CCEA include problem decomposition, collaborator selection, and individual fitness assignment. In this study, the SACC algorithm decomposes the original problem of (1) into three or six subpopulations according to the genotype settings of the GLSM. The original population and the subpopulations are represented as (8),

$$\begin{aligned}\mathbf{X} &= \{\mathbf{x}_i, i \in 1, 2, \ldots, N_g\} \\ &= \{x_{i,j}, i \in 1, 2, \ldots, N_g, j \in 1, 2, \ldots, N_p\} \\ &= \{x_j, j \in 1, 2, \ldots, N_p, N_g = 1\},\end{aligned} \tag{8}$$

where $\mathbf{x}_i$ denotes the subpopulation and N_g is the number of the subpopulations. x_j and $x_{i,j}$ denote the individual of the original population and subpopulation, respectively, N_p is the number of the individuals. The subpopulation is evolved by a genetic algorithm (GA). The best individual b_i of each subpopulation $\mathbf{x}_i$ are selected to form the complete solution $\mathbf{b} = \{b_i, i \in 1, 2, \ldots, N_g\}$ and the fitness of $\mathbf{b}$ is $f_b = \mathcal{F}(\mathbf{b})$, which is short for $f_b = \mathcal{F}(\mathbf{b}, \mathcal{D})$. The fitness of the individual $x_{i,j}$ is estimated by replacing the corresponding segment of complete solution $f_{i,j} = \mathcal{F}(x_{i,j}) = \mathcal{F}(\hat{\mathbf{b}}_{i,j})$, where $\hat{\mathbf{b}}_{i,j} = \{b_1, b_2, \ldots, x_{i,j}, \ldots, b_{N_g}\}$. The fitness of all individuals of subpopulation $\mathbf{x}_i$ is expressed as $\mathbf{f}_i = \mathcal{F}(\mathbf{x}_i) = \mathcal{F}(\hat{\mathbf{b}}_i)$.

The problem decomposition is based on the genotype settings, which is distinguished by different colors in Fig. 4(b)

with $N_g = 6$ and $N_g = 3$. The motifs $\{m_j^{l_1}, j = 1, 2, 3, 4\}$ are regarded as one subpopulation when $N_g = 3$. $N_g = 1$ means the population without decomposition, and the CCEA is therefore equivalent to the GA. An example of the fitness evaluation for the subpopulation $\mathbf{x}_3$ ($N_g = 6$) is also shown in Fig. 4. In $\hat{\mathbf{b}}_3 = \{\hat{\mathbf{b}}_{3,j}, j = 1, \ldots, N_p\}$, b_3 is replaced by $\{x_{3,j}, j = 1, \ldots, N_p\}$. Therefore, the fitness of $\mathbf{x}_3$ is $\mathbf{f}_3 = \mathcal{F}(\hat{\mathbf{b}}_3)$.

C. Random Forest Based Surrogate

To reduce the computation consumption, the individual predictor estimates the fitness of the individuals $\mathbf{x}_i$ by a random forest surrogate, which is represented as $\hat{\mathbf{f}}_i = \mathcal{RF}(\hat{\mathbf{b}}_i)$ [34], [56]. The random forest is an ensemble machine learning method by training numerous decision trees [35]. Each decision tree is trained with the data randomly sampled from the whole dataset. The mean and variance of the decision tree outputs are used as the predicted value and the uncertainty of the random forest output. The random forest performs better than other machine learning methods on discrete data. Since the genotype of the GLSM is hybrid coded, the random forest is well suited for establishing the surrogate of the GLSM.

The SACC algorithm employs $D = 100$ classification and regression trees (CARTs) [57] in the random forest regression method to create the surrogate. The surrogate employs the lower confidence bound (LCB) [58] described in (9) as the prediction value of $\hat{\mathbf{b}}_{i,j}$.

$$\mathcal{RF}(\hat{\mathbf{b}}_{i,j}) = \mu(\hat{\mathbf{b}}_{i,j}) - \rho \cdot \sigma(\hat{\mathbf{b}}_{i,j}), \tag{9}$$

where $\rho = 2.576$, $\mu(\hat{\mathbf{b}}_{i,j})$ and $\sigma(\hat{\mathbf{b}}_{i,j})$ are the mean and variance of the random forest output for $\hat{\mathbf{b}}_{i,j}$, respectively.

The details of the SACC algorithm are provided in Algorithm 3. In Algorithm 3, the SACC firstly samples k coordinate solutions $\mathbf{X^L} = \{x_j^L, j \in 1, 2, \ldots, k\}$ by Latin hypercube sampling (LHS) to create the random forest surrogate. During the evaluation process, $\mathbf{x}_i^O$ denotes the offspring individuals of the subpopulation. The SACC evaluates the most promising N_s offspring individuals $\mathbf{x}_i^S$ by the original function $\mathcal{F}$ in every η generations. Then the surrogate $\mathcal{RF}$ is updated according to the newly evaluated results, which can increase the prediction accuracy of $\mathcal{RF}$. The the best fitness f_b and individual $\mathbf{b}$ are then chosen from $\{f_b, \mathbf{f}_i^S\}$ and $\{\mathcal{F}(\mathbf{b}), \mathcal{F}(\hat{\mathbf{b}}_i^S)\}$, respectively. The corresponding recombination and mutation are employed depending on the coding type. The subpopulation of the next generation is selected from combining the parent and the offspring subpopulation $\{\mathbf{x}_i, \mathbf{x}_i^O\}$ using tournament selection.

Algorithm 3: SACC algorithm.

Input: The population size N_p, the number of evaluated individuals N_s, the evaluation interval η, and the number of coordinate solutions for LHS k.
Output: The best fitness f_b, the best solution $\mathbf{b}$.

```
Latin hypercube sampling X^L = {x_j^L, j ∈ 1, 2, ..., k};
Evaluate f^L = F(X^L);
Create the surrogate RF based on (f^L, X^L);
Update f_b ← min f^L, b ← arg min F(X^L);
for i ← 1 to N_g do
    Generate an initial subpopulation x_i;
    Estimate f̂_i = RF(b̂_i);
end
loop = 1;
while The termination is not satisfied do
    loop = loop + 1;
    for i ← 1 to N_g do
        Generate an offspring subpopulation x_i^O by the
          recombination and the mutation of x_i;
        Estimate f̂_i^O = RF(b̂_i^O);
        if loop == η then
            Select the most promising N_s offsprings x_i^S
              from x_i^O, where b̂_i^S = arg min RF(b̂_i^O);
            Evaluate f_i^S = F(b̂_i^S);
            Update the surrogate RF based on
              (f_i^S, b̂_i^S);
            Update f_b ← arg min {f_b, f_i^S},
              b ← arg min {F(b), F(b̂_i^S)};
            Replace the corresponding the evaluation
              values of x_i^S with f̂_i^S in f̂_i^O ;
            loop = 0;
        end
        Rank and select best N_p offspring based on
          arg min{f̂_i^O, f̂_i} from {x_i, x_i^O} as the
          subpolulation of the next generation x_i;
    end
end
return f_b, b;
```

VI. Experimental Results

This section firstly introduces the experimental settings in Section VI-A, and the overall results are then presented in Section VI-B. The ablation studies and the studies on the effectiveness of parallelism training are provided in Section VI-C and Section VI-D, respectively. Moreover, two additional experiments are provided in the Supplementary material. The architecture analysis illustrates the average path length and clustering coefficient are two effective architecture metrics to reflect the classification performance according to the results of Fig. S3. Finally, the sensitivity analysis results of the random forest surrogate, as shown in Fig. S4, suggest that $N_s = 2$ and $\eta = 10$ are the most appropriate values for the SACC. Note that all experimental results of the optimization process presented in this section are the average values of three independent runs.

A. Experimental Settings

The GLSM is simulated by a highly flexible and easily extensible spiking neural network simulator (Brian2) [59]. Brian2 is an ordinary differential equation solver based simulator for the time-driven or event-driven SNN simulation, which is more biologically plausible and hardware friendly than other matrix manipulation-based simulators. The simulation results can be

directly adopted for the neuromorphic hardware design. All of the following experiments are time-driven and simulated in the time step of 1 ms. The data parallelism training is implemented by Ray, which is a parallel distributed computation framework [37]. The experiments are simulated on the Ray cluster containing two X86-platform servers (Dell Precision 7920 Tower). Each sever consists of two CPUs (Intel Xeon Gold 5218R @ 2.10 GHz, 20 cores) and 96 GB physical RAM. In the cluster, two servers are connected by a network router for communication.

1) Datesets and Spatio-Temporal Coding: In this study, the proposed method is examined using three sequence classification benchmarks. This work employs two action recognition benchmarks (KTH [38] and HAR [39]) and a event-based vision [40] recognition benchmark called Neuromorphic-MNIST (N-MINIST) [41]. The KTH dataset is a video-based benchmark for detecting human behaviors. It contains 2391 video files of jogging, clapping, boxing, waving, and walking. Another human action recognition (HAR) benchmark is created from the inertial sensors of smartphones carried by 30 volunteers performing basic activities. N-MINIST is a spiking version of the MNIST dataset that uses an actuated pan-tilt dynamic vision sensor (DVS) camera to convert static images into spike neuromorphic vision.

The encoding layer converts the sequence data using spatio-temporal coding methods, which are believed to be more effective and biologically plausible than rate coding methods [53]. The KTH benchmark is encoded based on the difference between two time-adjacent video frames and the 5×5 max-pooling core [10]. Each video contains 160×120 pixels with an average length of 240 frames. The number of encoding neurons for KTH is 768. The KTH benchmark is separated into the training data, validation data and testing data with the ratio of $15:5:4$. The data for plasticity training randomly selects 20 percent of the training data.

The HAR benchmark converts the sensor signals into non-sequential data with 561 features. In this study, each feature is transformed into the spike train with 10 ms duration time by three square cosine encoding neurons [10], [60]. Thus, the total number of the encoding neuron is 1683. HAR benchmark contains 7352 training data and 2947 testing data. The validation data and plasticity training data randomly select 30 percent and 20 percent of the training data, respectively.

The data of the N-MINIST benchmark is comprised of event-based spikes produced by the DVS camera. The encoding neuron needs to convert the event-based spikes into the spike trains with 2312 features and a duration of 10 ms. This work employs the conversion technique mentioned in SpikingJelly [61]. The 3000 training and 500 testing data are randomly selected from 60000 original training data and 10000 original testing data, respectively. Additional 500 validation data and 120 plasticity training data are randomly selected from the original training data.

2) Algorithm and Parameter Settings: Most parameters of the GLSM are automatically selected by the optimization algorithms of ENAS. Other constant parameter settings are listed in Table II.

TABLE II
THE CONSTANT PARAMETER SETTINGS OF THE GLSM

Parameter		Constant value
increasing incremental coefficient	A_P	0.01
decreasing incremental coefficient	A_D	0.0105
upper bound of synaptic strength	w_{ij}^{max}	1
lower bound of synaptic strength	w_{ij}^{min}	0
rest membrane potential	u_{rest}	0.2
basic neural threshold	T_0	0.2
neural threshold increment	ΔT	0.01
membrane time decay	τ_n	$100ms$
refractory period	t_{ref}	$2ms$

TABLE III
THE COMPARISON OF THE CLASSIFICATION ACCURACIES FOR VARIOUS NEURAL ARCHITECTURES AND OPTIMIZATION METHODS ON THE N-MNIST, KTH AND HAR BENCHMARKS

	Method	N-MNIST	KTH	HAR
SOTA	BCM-GRN [63]	-	88.15%	-
	STDP-LSM [9]	-	66.70%	-
	NAS-LSM [23]	92.50%	-	-
	CMAES-GP-LSM [10]	91.80%	92.08%	93.85%
GA	GA-GLSM	91.40%	88.51%	91.59%
	GA-FI-GLSM	90.20%	88.25%	89.82%
	SAGA-RFE-GLSM	91.40%	90.34%	91.46%
	SAGA-RF-GLSM	91.60%	90.08%	91.99%
CCEA	CCEA-GLSM	91.80%	**92.59%**	93.46%
	CCEA-FI-GLSM	91.40%	91.91%	91.92%
	SACC-RFE-GLSM	92.20%	91.38%	92.72%
	SACC-RF-GLSM	**92.80%**	92.43%	**94.06%**

In the following experiments, we optimize the GLSM primarily based on the CCEA and SSAC. The GA is tested as a comparison without problem decomposition. Apart from the classical random forest surrogate (named RF for short), the effective and efficient end-to-end performance predictor (E2EPP) proposed in [34] is also compared as the random forest surrogate (named RFE for short). To verify the superiority of the surrogate, a fitness inheritance algorithm [62] is also examined as the individual fitness assignment of the CCEA.

In this work, the CCEA decomposes the population into three or six subpopulations. The number of offspring individuals is $N_g = 10$ for the CCEA and GA. The RF and RFE employ $D = 100$ CARTs, with $k = 100$ initial coordinate solutions for LHS. The surrogate assisted GA (SAGA) and SACC algorithm evaluate $N_s = 2$ individuals out of the $N_g = 20$ offspring individuals between each $\eta = 10$ generations. The FI estimates half of the offspring individual fitness based on the inherited fitness in each generation. The termination condition for the comparison of the algorithms is 600 evaluations of the original function $\mathcal{F}(x, \mathcal{D})$.

B. Overall Results

The classification accuracies on the N-MNIST, KTH and HAR benchmarks are presented in Table III. The first four rows of Table III present the state-of-the-art (SOTA) performance of SNNs and LSMs trained by plasticity rules or architecture optimization algorithms. The four SOTA studies are the multi-layer

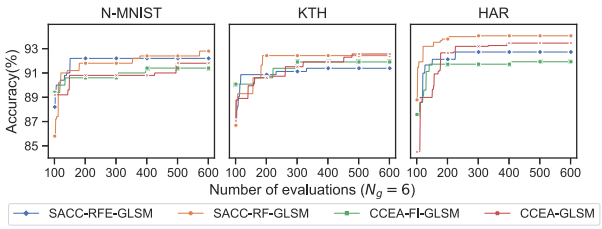


Fig. 5. The evolutionary processes of CCEA-GLSM, CCEA-FI-GLSM, SACC-RFE-GLSM and SACC-RF-GLSM for N-MINST, KTH and HAR benchmarks. All algorithms are evolved with $N_g = 6$ subpopulations.

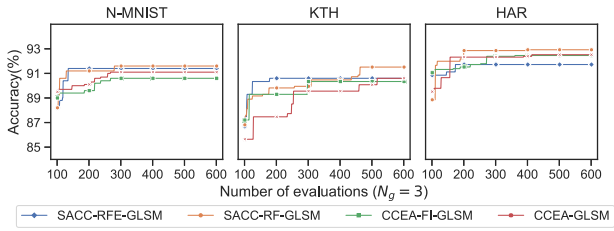


Fig. 6. The evolutionary processes of CCEA-GLSM, CCEA-FI-GLSM, SACC-RFE-GLSM and SACC-RF-GLSM for N-MINST, KTH and HAR benchmarks. All algorithms are evolved with $N_g = 3$ subpopulations.

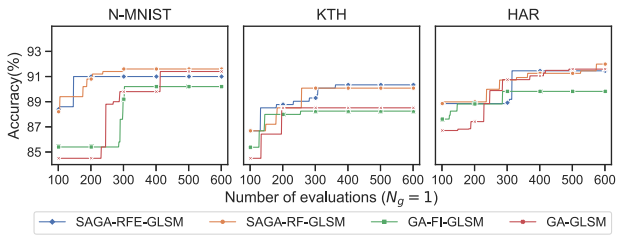


Fig. 7. The evolutionary processes of GA-GLSM, GA-FI-GLSM, SAGA-RFE-GLSM and SAGA-RF-GLSM for N-MINST, KTH and HAR benchmarks. The algorithms are evolved with $N_g = 1$, which means no decomposition is adopted.

SNN trained by the BCM plasticity rule and a gene regulatory network [63], the LSM trained by STDP [9], the NAS based LSM [23] and the LSM optimized by Gaussian process (GP) surrogate assisted covariance matrix adaptation evolution strategy (CMA-ES) [10], which are denoted as BCM-GRN, STDP-LSM, NAS-LSM and CMAES-GP-LSM, respectively. To the best of our knowledge, these represent the state-of-the-art LSM models. BCM-GRN and STDP-LSM are the baselines of the KTH benchmark using SNNs as the classification model. NAS-LSM and CMAES-GP-LSM are the most recent and relevant research on architecture optimization of LSMs. The following eight rows show the performance of the GLSM optimized by the evolutionary algorithms (GA or CCEA) with different fitness assignment strategies (RF, RFE or FI). The symbol "-" means the corresponding results cannot be retrieved from the published papers or assessed from the corresponding public programs.

As shown in Table III, CCEA-GLSM and SACC-RF-GLSM achieve the best performance on the KTH, HAR and N-MNIST benchmarks. Generally, the CCEA and SACC optimized GLSM outperform the existing SOTA methods on all three benchmarks. Compared with the GA, the CCEA performs better in all of 12 comparisons. The results prove that the co-evolutionary strategy provides a positive effect for the architecture optimization of the GLSM. Compared with different fitness assignment strategies, the accuracies of GA-GLSM and CCEA-GLSM are generally higher than GA-FI-GLSM and CCEA-FI-GLSM on all of the three benchmarks. The algorithms using the RF and RFE also achieve better performance than the algorithms based on FI for both the CCEA and GA. Those results suggest that the surrogate-assisted algorithms are more effective than FI-based algorithms for fitness assignment strategies of ENAS. The CCEA is evenly matched with the SACC algorithm in the performance comparisons. SACC-RF-GLSM achieves the best accuracy on N-MINIST and HAR while only slightly lower than CCEA-GLSM by 0.15% on KTH. For a fair comparison, all the surrogates are updated online during the optimization process. The RFE is actually designed for the off-line scenario, which makes their accuracies slightly poorer than those of the RF.

This section also demonstrates the evolutionary processes of the optimization methods for the GLSM in Table III. Figs. 5 and 6 display the evolutionary process of the CCEA-based algorithms on the NMINST, KTH and HAR benchmarks. The evolutionary processes of the GA-based algorithms are shown in Fig. 7.

To compare the co-evolutionary strategy with different subpopulations, this work utilizes the population decomposition strategies with subpopulations of $N_g = 3$ and $N_g = 6$ for CCEA-based algorithms. According to the results in Figs. 5, 6, and 7, it is apparent that the performance of $N_g = 6$ is better than $N_g = 3$ and the GA ($N_g = 1$) on all the three benchmarks. Additionally, surrogate-assisted methods, such as SACC-RF-GLSM and SAGA-RF-GLSM, outperform methods without surrogates, with only the performance of SACC-RF-GLSM being slightly lower than CCEA-GLSM on the KTH benchmark. The results indicate that surrogates can help evolutionary algorithms in quickly finding better solutions during the early stage of the evolution process, whereas the algorithm without surrogate will theoretically perform better in the settings if it has infinite computational resources. Therefore, the surrogate-assisted optimization algorithms are well suited for the ENAS-based GLSM.

In summary, these results indicate that the proposed GLSM architecture has numerous advantages over the existing LSM architectures. The problem decomposition strategy of the CCEA and the RF surrogate are particularly beneficial for the architecture evolution of the GLSM.

C. Ablation Studies for Network Structure and Plasticity

To investigate the effect of the complex network models and the plasticity rules, this subsection assesses the performance of the GLSM without plasticity training and the GLSM with only random networks as the ablation experiments. The ablation experiments are assessed by SACC-RF-GLSM with $N_g = 6$ subpopulations on the three benchmarks. The evolutionary processes are presented in Fig. 8. Without plasticity training, SACC-RF-GLSM has a significant weakness in improving the classification accuracy. In addition, the performance of the GLSM with only random networks is also worse than the GLSM with the four network models.

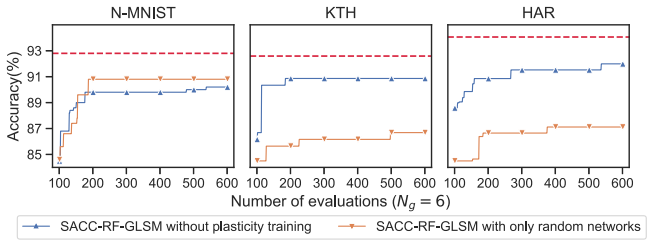


Fig. 8. The ablation experiments of the complex network models and the plasticity for GLSM tested with SACC-RF-GLSM ($N_g = 6$) on N-MINST, KTH and HAR benchmarks. The dash lines are the reference lines of the best results in Table III.

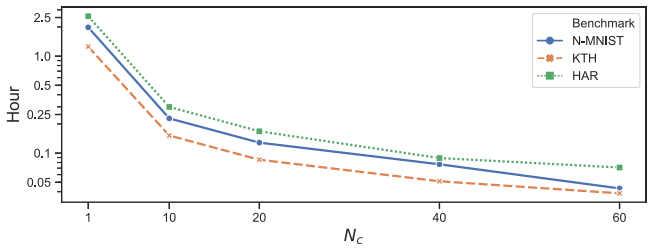


Fig. 9. The time consumption of a single GLSM evaluation with different processors.

Overall, these results imply that the plasticity rules and the complex network models can enhance the classification capacity of the ENAS-based GLSM. Additionally, the genotype contains only the parameters of the plasticity rules, rather than the individual values, reducing the dimension of the search space.

D. Effectiveness of Data Parallelism Training

This section compares the time consumption of a single GLSM evaluation with different numbers of processors to highlight the effectiveness of the data parallelism. The time consumption is evaluated by the processors $N_c = 1, 10, 20, 40, 60$ in the Ray cluster, respectively. The results are shown in Fig. 9, and the ordinate of the figure is shown on a logarithmic scale. The basic time consumption with only one process is approximate two hours and the whole architecture search takes approximate $2 \times 600 = 1200$ hours. The huge time consumption of the baseline approach makes it prohibitive to explore more complex architectures. With an increased number of the processors, it can be seen that the time consumption decreases significantly. The time consumption of one GLSM evaluation with 60 processors is around 0.07 h and the total architecture search takes only $0.07 \times 600 = 42$ hours. This clearly indicates that the data parallelism remarkably speeds up the evaluation of the GLSM, making it possible to explore more effective architecture of the GLSM in the future.

VII. Conclusion and Future Work

This paper proposed a surrogate-assisted CCEA for evolving large-scale modular reservoir for liquid state machines, called SACC-GLSM. A block-based hierarchical representation with complex network models and plasticity rules is employed to explore more effective network structures. To avoid high-dimensional search space, the optimization strategy decomposes the original optimization problem into numerous low-dimensional subproblems using the CCEA. Furthermore, we reduce the computational cost by estimating the fitness of coordinate solutions using a random forest surrogate as the fitness assignment strategy. Finally, the data parallelism is also adopted to speed up the plasticity training and the state extraction process. The proposed method is verified on the N-MNIST, KTH and HAR benchmarks. The experimental results confirm that the proposed SACC-GLSM achieves state-of-the-art performance on the three classification benchmarks. Moreover, employing plasticity rules and various complex network models can significantly improve classification performance. Our experimental results also demonstrate that the proposed problem decomposition strategy and the random forest surrogate are effective for the ENAS-based GLSM.

Nevertheless, a few challenges remain open. First, the proposed block-based hierarchical representation only employs four motifs in each level, which is constrained by limited computational resources. The number of motif types can be increased in the future and more complex network structures can be generated. Additionally, the modular architecture provides the advantage of concurrently processing multiple types of input stimuli. After being trained by synaptic plasticity rules, the modules of LSMs can exhibit various sensitivities to different features of the input stimuli, which can help reduce module interference. Therefore, another future work is to investigate the robustness and structural properties of the modular LSMs for multi-task classification. Finally, the proposed GLSM simulates basic spiking neurons with X86-platform CPUs, resulting in a lower computing efficiency than deep learning models. However, after being optimized using the SACC algorithm on the X86 platform, the GLSM may be used in specially designed neuromorphic hardware and various low-power devices, allowing it to cover a wider range of application scenarios than deep learning.

References

[1] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.
[2] Q. Yu, H. Tang, J. Hu, and K. C. Tan, *Neuromorphic Cognitive Systems*. Berlin, Germany: Springer, 2017.
[3] J. Pei et al., "Towards artificial general intelligence with hybrid Tianjic chip architecture," *Nature*, vol. 572, no. 7767, pp. 106–111, 2019.
[4] Y. Zhang et al., "A system hierarchy for brain-inspired computing," *Nature*, vol. 586, no. 7829, pp. 378–384, 2020.
[5] M. Nadji-Tehrani and A. Eslami, "A brain-inspired framework for evolutionary artificial general intelligence," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 12, pp. 5257–5271, Dec. 2020.
[6] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Netw.*, vol. 10, no. 9, pp. 1659–1671, 1997.
[7] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Comput.*, vol. 14, no. 11, pp. 2531–2560, 2002.
[8] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. M. McGinnity, "A review of learning in biologically plausible spiking neural networks," *Neural Netw.*, vol. 122, pp. 253–272, 2020.
[9] J. Chrol-Cannon and Y. Jin, "Learning structure of sensory inputs with synaptic plasticity leads to interference," *Front. Comput. Neurosci.*, vol. 9, 2015, Art. no. 103.
[10] Y. Zhou, Y. Jin, and J. Ding, "Surrogate -assisted evolutionary search of spiking neural architectures in liquid state machines," *Neurocomputing*, vol. 406, pp. 12–23, 2020.

[11] G. Bellec et al., "A solution to the learning dilemma for recurrent networks of spiking neurons," *Nature Commun.*, vol. 11, no. 1, 2020, Art. no. 3625.

[12] X. Wang, X. Lin, and X. Dang, "Supervised learning in spiking neural networks: A review of algorithms and evaluations," *Neural Netw.*, vol. 125, pp. 258–280, 2020.

[13] Q. Yu, H. Li, and K. C. Tan, "Spike timing or rate? Neurons learn to make decisions for both through threshold-driven plasticity," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2178–2189, Jun. 2019.

[14] D. O. Hebb, *The Organization of Behavior: A Neuropsychological Theory*. London, U.K.: Psychology Press, 2005.

[15] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2635–2649, Nov. 2015.

[16] S. Song, K. D. Miller, and L. F. Abbott, "Competitive Hebbian learning through spike-timing-dependent synaptic plasticity," *Nature Neurosci*, vol. 3, no. 9, pp. 919–926, 2000.

[17] P. Diehl and M. Cook, "Unsupervised learning of digit recognition using spike-timing-dependent plasticity," *Front. Comput. Neurosci.*, vol. 9, 2015, Art. no. 99.

[18] N. Soures and D. Kudithipudi, "Deep liquid state machines with neural plasticity for video activity recognition," *Front. Neurosci.*, vol. 13, 2019, Art. no. 686.

[19] M. Bear, B. Connors, and M. A. Paradiso, *Neuroscience: Exploring the Brain*., Boston, MA, USA: Jones & Bartlett2020.

[20] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[21] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, "A survey on evolutionary neural architecture search," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–21, 2021.

[22] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German Nat. Res. Center Inf. Technol. GMD Tech. Rep.*, vol. 148, 2001, Art. no. 34.

[23] S. Tian, L. Qu, L. Wang, K. Hu, N. Li, and W. Xu, "A neural architecture search based framework for liquid state machine design," *Neurocomputing*, vol. 443, pp. 174–182, 2021.

[24] M. Newman, *Networks: An Introduction*. London, U.K.: Oxford Univ. Press, 2010.

[25] Y. Okumura and N. Wakamiya, "Analysis of reservoir structure contributing to robustness against structural failure of liquid state machine," in *Proc. Int. Conf. Artif. Neural Netw.*, 2020, pp. 435–446.

[26] T. Sakaguchi and N. Wakamiya, "Consideration on liquid structure contributing to discrimination capability of liquid state machine," *Nonlinear Theory Appl.*, vol. 11, no. 1, pp. 36–59, 2020.

[27] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1284–1293.

[28] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BJQRKzbA-

[29] X. Wang, Y. Jin, and K. Hao, "Echo state networks regulated by local intrinsic plasticity rules for regression," *Neurocomputing*, vol. 351, pp. 111–122, 2019.

[30] M. C. W. V. Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable Hebbian learning from spike timing-dependent plasticity," *J. Neurosci.*, vol. 20, no. 23, pp. 8812–8821, 2000.

[31] X. Ma et al., "A survey on cooperative co-evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 23, no. 3, pp. 421–441, Jun. 2019.

[32] S. S. Tirumala, "Evolving deep neural networks using coevolutionary algorithms with multi-population strategy," *Neural Comput. Appl.*, vol. 32, no. 16, pp. 13051–13064, 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-04749-2

[33] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez, "COVNET: A cooperative coevolutionary model for evolving artificial neural networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 3, pp. 575–596, May 2003.

[34] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 350–364, Apr. 2020.

[35] T. K. Ho, "Random decision forests," in *Proc. IEEE 3rd Int. Conf. Document Anal. Recognit.*, 1995, vol. 1, pp. 278–282.

[36] H. Zhu, H. Zhang, and Y. Jin, "From federated learning to federated neural architecture search: A survey," *Complex Intell. Syst.*, vol. 7, no. 2, pp. 639–657, 2021.

[37] P. Moritz et al., "Ray: A distributed framework for emerging AI applications," in *Proc. 13th USENIX Symp. Operating Syst. Des. Implementation*, 2018, pp. 561–577.

[38] C. Schuldt, I. Laptev, and B. Caputo, "Recognizing human actions: A local SVM approach," in *Proc. IEEE 17th Int. Conf. Pattern Recognit.*, 2004, vol. 3, pp. 32–36.

[39] J.-L. Reyes-Ortiz, L. Oneto, A. Samà, X. Parra, and D. Anguita, "Transition-aware human activity recognition using smartphones," *Neurocomputing*, vol. 171, pp. 754–767, 2016.

[40] G. Gallego et al., "Event-based vision: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 1, pp. 154–180, Jan. 2022.

[41] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Front. Neurosci.*, vol. 9, 2015, Art. no. 437.

[42] J. You, J. Leskovec, K. He, and S. Xie, "Graph structure of neural networks," in *Proc. 37th Int. Conf. Mach. Learn.*, H. D. III and A. Singh, Eds., 2020, vol. 119, pp. 10881–10891.

[43] N. Rodriguez, E. Izquierdo, and Y.-Y. Ahn, "Optimal modularity and memory capacity of neural reservoirs," *Netw. Neurosci.*, vol. 3, no. 2, pp. 551–566, 2019.

[44] C. Gallicchio, A. Micheli, and L. Pedrelli, "Design of deep echo state networks," *Neural Netw.*, vol. 108, pp. 33–47, 2018.

[45] Z. Deng and Y. Zhang, "Collective behavior of a small-world recurrent neural system with scale-free distribution," *IEEE Trans. Neural Netw.*, vol. 18, no. 5, pp. 1364–1375, Sep. 2007.

[46] Y. Xu et al., "Renas: Relativistic evaluation of neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 4411–4420.

[47] H. Tan et al., "RelativeNAS: Relative neural architecture search via slow-fast learning," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–15, 2021.

[48] H. Zhang, Y. Jin, R. Cheng, and K. Hao, "Efficient evolutionary search of attention convolutional networks via sampled training and node inheritance," *IEEE Trans. Evol. Comput.*, vol. 25, no. 2, pp. 371–385, Apr. 2021.

[49] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 1, pp. 1997–2017, 2019.

[50] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74, no. 1, pp. 47–97, 2002.

[51] B. Bollobás, C. Borgs, J. T. Chayes, and O. Riordan, "Directed scale-free graphs," in *Proc. 14th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2003, vol. 3, pp. 132–139.

[52] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM J. Comput.*, vol. 1, no. 2, pp. 146–160, 1972.

[53] W. Gerstner and W. M. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge, U.K.: Cambridge Univ. Press, 2002.

[54] X. Peng, Y. Jin, and H. Wang, "Multimodal optimization enhanced cooperative coevolution for large-scale optimization," *IEEE Trans. Cybern.*, vol. 49, no. 9, pp. 3507–3520, Sep. 2019.

[55] P. Yang, K. Tang, and X. Yao, "Turning high-dimensional optimization into computationally expensive optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 143–156, Feb. 2018.

[56] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm Evol. Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[57] W.-Y. Loh, "Classification and regression trees," *WIREs Data Mining Knowl. Discov.*, vol. 1, no. 1, pp. 14–23, 2011.

[58] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. D. Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, Jan. 2016.

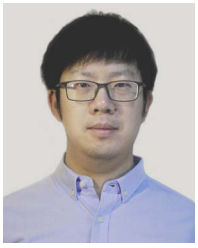
[59] M. Stimberg, D. Goodman, V. Benichoux, and R. Brette, "Equation-oriented specification of neural models for simulations," *Front. Neuroinform.*, vol. 8, 2014, Art. no. 6.

[60] Q. Wu, T. M. McGinnity, L. P. Maguire, B. Glackin, and A. Belatreche, "Learning under weight constraints in networks of temporal encoding spiking neurons," *Neurocomputing*, vol. 69, no. 16, pp. 1912–1922, 2006.

[61] W. Fang et al., "Spikingjelly," 2020. [Online]. Available: https://github.com/fangwei123456/spikingjelly

[62] A. Hameed, D. Corne, D. Morgan, and A. Waldock, "Large-scale optimization: Are co-operative co-evolution and fitness inheritance additive?," in *Proc. 13th U.K. Workshop Comput. Intell.*, 2013, pp. 104–111.

[63] Y. Meng, Y. Jin, and J. Yin, "Modeling activity-dependent plasticity in BCM spiking neural networks with application to human behavior recognition," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 1952–1966, Dec. 2011.



Yan Zhou received the bachelor's and master's degrees in 2011 and 2015, respectively, from Northeastern University, Shenyang, China, where he is currently working toward the Ph.D. degree with the State Key Laboratory of Synthetical Automation for Process Industries. His research interests include brain-inspired computing, reservoir computing, evolutionary neural architecture search and surrogate-assisted evolutionary optimization.

Yaochu Jin (Fellow, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in automatic control from Zhejiang University, Hangzhou, China, in 1988, 1991, and 1996, respectively, and the Dr.-Ing. degree in neuroinformatics from Ruhr University Bochum, Bochum, Germany, in 2001. He is an Alexander von Humboldt Professor for artificial intelligence, with the Faculty of Technology, Bielefeld University, Bielefeld, Germany, and a Distinguished Chair Professor of Computational Intelligence with the Department of Computer Science, University of Surrey, Guildford, U.K. He was a Finland Distinguished Professor with the University of Jyväskylä, Jyväskylä, Finland, and Changjiang Distinguished Visiting Professor with Northeastern University, Shenyang, China. He was an IEEE Distinguished Lecturer from 2013 to 2015 and from 2017 to 2019. His main research interests include evolutionary optimization and learning, deep learning and optimization, trustworthy machine learning, and evolutionary developmental systems. Dr. Jin was the recipient of the 2014, 2016, and 2019 IEEE Computational Intelligence Magazine Outstanding Paper Award, the 2018 and 2020 IEEE Transactions on Evolutionary Computation Outstanding Paper Award, and the Best Paper Award of the 2010 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology. He was named Highly Cited Researcher by the Web of Science Group from 2019 to 2022. He is currently an Editor-in-Chief of Complex & Intelligent Systems. He is a Member of Academia Europaea.

Yao Sun received the B.Sc. and M.Sc. degrees in automation from Northeastern University, Shenyang, China, in 2011 and 2013, respectively, and the Ph.D. degree in information and communication engineering from The University of Tokyo, Tokyo, Japan, in 2019. He is currently an Assistant Researcher with Shenyang Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include reinforcement learning, and intelligent decision-making.

Jinliang Ding (Senior Member, IEEE) received the bachelor's, master's, and Ph.D. degrees in control theory and control engineering from Northeastern University, Shenyang, China, respectively. He is currently a Professor of the State Key Laboratory of Synthetical Automation for Process Industry, Northeastern University. He has authored or coauthored more than 200 refereed journal papers and refereed papers at international conferences. He has also invented or coinvented over 50 patents. His research interests include modeling, plant-wide control and optimization for the complex industrial systems, machine learning, industrial artificial intelligence, and computational intelligence and application. Dr. Ding was the recipient of the Young Scholars Science and Technology Award of China in 2016, the National Science Fund for Distinguished Young Scholars in 2015, the National Technological Invention Award in 2013, and three First-Prize of Science and Technology Awards of the Ministry of Education in 2006, 2012, and 2018, respectively. One of his articles published on Control Engineering Practice was selected for the Best Paper Award of 2011–2013.