Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

A memristive all-inclusive hypernetwork for parallel analog deployment of full search space architectures

Bo Lyu [a], Yin Yang [b], Yuting Cao [b], Tuo Shi [a], Yiran Chen [c], Tingwen Huang [d], Shiping Wen [e,*]

[a] *Zhejiang Lab, Hangzhou, Zhejiang, China*
[b] *College of Science and Engineering, Hamad Bin Khalifa University, Doha 5855, Qatar*
[c] *Department of Electrical and Computer Engineering, Duke University, Durham, NC, USA*
[d] *Science Program, Texas A & M University at Qatar, Doha 23874, Qatar*
[e] *Australian AI Institute, Faculty of Engineering and Information Technology, University of Technology Sydney, NSW 2007, Australia*

ARTICLE INFO

Keywords:
Memristor
Convolutional neural network
Neural architecture search
Hypernetwork

ABSTRACT

In recent years, there has been a significant advancement in memristor-based neural networks, positioning them as a pivotal processing-in-memory deployment architecture for a wide array of deep learning applications. Within this realm of progress, the emerging parallel analog memristive platforms are prominent for their ability to generate multiple feature maps in a single processing cycle. However, a notable limitation is that they are specifically tailored for neural networks with fixed structures. As an orthogonal direction, recent research reveals that neural architecture should be specialized for tasks and deployment platforms. Building upon this, the neural architecture search (NAS) methods effectively explore promising architectures in a large design space. However, these NAS-based architectures are generally heterogeneous and diversified, making it challenging for deployment on current single-prototype, customized, parallel analog memristive hardware circuits. Therefore, investigating memristive analog deployment that overrides the full search space is a promising and challenging problem. Inspired by this, and beginning with the *DARTS* search space, we study the memristive hardware design of primitive operations and propose the memristive all-inclusive hypernetwork that covers 2×10^{25} network architectures. Our computational simulation results on 3 representative architectures (*DARTS-V1*, *DARTS-V2*, *PDARTS*) show that our memristive all-inclusive hypernetwork achieves promising results on the CIFAR10 dataset (89.2% of *PDARTS* with 8-bit quantization precision), and is compatible with all architectures in the *DARTS* full-space. The hardware performance simulation indicates that the memristive all-inclusive hypernetwork costs slightly more resource consumption (nearly the same in power, $22\% \sim 25\%$ increase in $\texttt{Latency}$, $1.5\times$ in $\texttt{Area}$) relative to the individual deployment, which is reasonable and may reach a tolerable trade-off deployment scheme for industrial scenarios.

1. Introduction

The rapid development of neural networks benefits from the fetching of big data and the development of large-scale computing power. However, the state-of-the-art deep learning models make progress at the cost of ever-increasing computational resource consumption and memory access demand. With the failure of Moore's Law and the limited development of memory technology (high memory access efficiency and storage density cannot be achieved simultaneously), the constrained data throughput between the processing units (PUs) and the memory devices has become one of the most critical performance and energy bottlenecks in all kinds of computing architectures (Boroumand et al., 2018). To mitigate this "Memory Wall" issue, some application-specific integrated circuits (ASICs) adopt large-scale on-chip memory to store the synaptic weights of neural networks. For instance, to promote the data bandwidth and locality, the series of DianNao (Chen, Du, et al., 2014; Chen, Luo, et al., 2014) place eDRAMs on the chip, enabling near-memory computing. In contrast, processing-in-memory (PIM)/computing-in-memory (CIM) presents a solution that directly integrates the data storage and computation (Chi et al., 2016). Memristor, the fourth fundamental passive circuit element, was theoretically predicted by Chua (1971) in the 1970s. Since the first physical realization (a *TiO2* thin-film structure) by HP Lab in 2008 (Strukov, Snider, Stewart, & Williams, 2008), it has attracted

* Corresponding author.
E-mail addresses: bo.lyu@zhejianglab.com (B. Lyu), yyang@hbku.edu.qa (Y. Yang), ycao@hbku.edu.qa (Y. Cao), shituo@zhejianglab.com (T. Shi), yiran.chen@duke.edu (Y. Chen), tingwen.huang@qatar.tamu.edu (T. Huang), shiping.wen@uts.edu.au (S. Wen).

https://doi.org/10.1016/j.neunet.2024.106312
Received 17 October 2023; Received in revised form 28 January 2024; Accepted 8 April 2024
Available online 15 April 2024

a lot of attention and research from academia and industry (Hu, Li, Wu, & Rose, 2012; Kim, Zhang, & Li, 2015; Li et al., 2013), and is considered to be the most ideal logical computing unit in neuromorphic computing architectures. The processing-in-memory platforms based on the memristor crossbar reduced the time complexity of matrix–vector multiplication from $O(n^2)$ to $O(1)$, thereby significantly optimizing the computing of specific network models. While most PIM-based platforms (Chi et al., 2016; Shafiee et al., 2016; Song, Zhuo, Qian, Li, & Chen, 2018; Yang, Yan, Li, & Chen, 2020; Yang, Yan, Li, Kwon, et al., 2020) consist of analog and digital components, with the memristive crossbars as the analog components, and most other components being digital, such as controllers, buffers, etc (Krestinskaya, James, & Chua, 2019). Some other works study the memristive crossbar-based full parallel analog computing platforms (Wen et al., 2021; Wen, Xie, Yan, Huang, & Zeng, 2018; Yakopcic, Alom, & Taha, 2016, 2017), which take full advantage of the parallel analog processing capabilities of memristive crossbars to achieve multiple output feature maps of CNNs in a single processing cycle. However, these parallel analog memristive platforms are regular and structured, rather than heterogeneous and diversified, posing challenges for the parallel analog memristive circuit design compatibility with various neural network architectures. Specifically, this circuit modeling approach necessitates that each neural network architecture correspond to a unique circuit design or computational engine, leading to hardware designs that are heterogeneously diverse. Such heterogeneity and diversity cause incompatibility with different algorithmic applications, contrasting with the reconfigurable nature of digital computing and memory integrated circuits. Therefore, even though re-programming can change the synaptic weights, the application scenarios of these platforms are also limited.

On the other side, to relieve the burdensome workload and knowledge dependency of the human expert from the network design works (Zoph & Le, 2017), NAS has become a mature automated machine learning (AutoML) technology (He, Zhao, & Chu, 2021) and has surpassed the manually designed state-of-the-art networks in various tasks (Loni, Mohan, Asadi, & Lindauer, 2023). Therefore, it is critical to explore the design of NAS-based network architectures on parallel analog memristive platforms, aiming for full-space compatibility. Addressing this involves navigating some remarkable challenges:

(1) While NAS-based architectures are formed through a structured stack of cells or blocks (Liu, Simonyan, & Yang, 2019; Zoph & Le, 2017), a distinctive challenge arises from the nature of connections within and between these cells, which differ significantly from those in manually crafted networks. Also, the integration pattern of features in NAS networks is both dense and diverse, leading to considerable difficulties in designing parallel analog memristive circuits.
(2) The compatibility issue, *i.e.*, memristive neural networks need to be compatible with all architectures in the NAS search space. This implies that any architecture could be deployed on memristive circuit hardware solely through the re-programming.

Motivated by this, this work builds upon the previous parallel analog memristive platforms (Wen et al., 2021; Yakopcic et al., 2016, 2017), but extends to the compatible modeling of full search space (NAS-based) architectures deployment, with the consideration of the characteristics of memristive circuits, efficiency, and the enhanced compatibility and adaptability to various computational tasks.

Overall, the contributions of this article can be summarized as follows:

(1) We first identify and address the gap between NAS-based networks and architecture-specialized memristive parallel analog circuit design, which is crucial for the application of memristive neural networks in practical scenarios.
(2) We model all the memristor-based circuits of the primitive operation in *DARTS* (Liu et al., 2019) search space, thereby constructing the memristive all-inclusive hypernetwork to meet the compatibility requirement of the NAS-based architecture.
(3) The full-space compatible memristive deployment framework presents a versatile approach that may be extended to other analog PIM platforms, *e.g.*, phase-change memory (PCM). It may also support the candidate architecture deployment to obtain resource-aware metrics, which are instrumental in serving as a part of the hardware-aware NAS pipeline.

Our computation simulation experiments on 3 representative architectures (*DARTS-V1*, *DARTS-V2*, *PDARTS* (Chen, Xie, Wu, & Tian, 2019)) demonstrate promising results on the CIFAR10 dataset achieved without the need for re-design of the hardware circuit, but rather merely by the re-programming of architecture encoding matrix on the memristive hypernetwork. The hardware performance simulation indicates that the memristive all-inclusive hypernetwork (*Hyper-DARTS-V1*, *Hyper-DARTS-V2*) incurs slightly more resource consumption (power, energy, latency) compared to individual deployment, a trade-off that is reasonably and tolerably acceptable for industrial scenarios.

2. Related work

Processing-in-memory (PIM) and neuromorphic computing. Current advances in microprocessors and memory technologies present several serious obstacles to the traditional von-Neumann architectures, namely the stalled single-thread performance, the limited data throughput, and constrained energy efficiency. Importantly, research reveals that the data exchange between Processing Units (PUs) and off-chip storage devices (hard disks, flashes) consumes two orders of magnitude more energy than a floating-point operation (Keckler, Dally, Khailany, Garland, & Glasco, 2011). This is particularly critical in terms of neural network applications, which highly rely on data throughput (both with respect to feature maps and weight parameters). The "Memory Wall" problem of von-Neumann architecture is therefore a significant bottleneck. As an emerging technology, processing in/near memory engines (Ahn, Hong, Yoo, Mutlu, & Choi, 2015) provide a promising solution to address these challenges, owing to their capability to achieve low-power consumption and high inference efficiency. Through a parameterized analytical model, the Bitlet (Ronen et al., 2022) presents a feasible way to compare Processing-in-Memory (PIM) and CPU systems in terms of throughput, power, and energy. In recent years, processing-in-memory (PIM) architectures have been widely studied for different neural network applications (Chi et al., 2016; Merolla et al., 2011; Shafiee et al., 2016; Song et al., 2018), in which computing architectures are specifically optimized based on the requirements of the algorithm and deep learning models. Furthermore, in Dazzi, Sebastian, Parnell, Francese, Benini, and Eleftheriou (2021), an innovative efficient communication fabric based on graph homomorphism verification is introduced, for the mapping of mainstream CNNs on in-memory computing cores.

Memristor crossbar based analog computing. Memristive crossbar arrays are typically constructed by word lines (WLs) and bit lines (BLs) with memristors that are placed orthogonal to each other. These arrays, with the help of peripheral circuits, not only store synaptic weights but also facilitate PIM operations, including matrix–vector multiplication. Research in this area focuses on two main aspects: Some works Wen, Hu, Yang, Huang, Zeng, and Song (2019), Wen et al. (2021), Yakopcic et al. (2017), Yang, Yan, Li, and Chen (2020) concentrate on the implementation (ex-situ) of different types of neural networks on memristor crossbars. Conversely, other works study the training procedure of neural networks (in-situ) (Cheng et al., 2017; Li, Wang, Wang, Chen, & Yang, 2014; Yan et al., 2020). For example, Wen, Hu, et al. (2019) propose a novel memristor-based computational architecture for Echo State Network (ESN) with the online Least Mean Square (LMS) algorithm. Additionally, other works (Wen et al., 2021; Wen, Wei, Zeng, & Huang, 2018) focus on the implementation and acceleration of RNN/LSTM on memristor crossbar based parallel analog platforms. In the field of image processing, targeting classification and segmentation tasks, some works (Lyu, Hamdi, et al., 2023; Lyu, Wang, et al., 2023) also present effective strategies to accelerate and save resource consumption of memristor-based computing in CNNs. Ultimately, the memristor-based PIM chip is anticipated to offer a non-von-Neumann hardware solution for the forthcoming emerging neural network applications on edge computing platforms.

Neural architecture search and model compression. With the rapid development of deep learning technology in recent years, its associated design and hyperparameter tuning processes have become extremely tedious for researchers and engineers. To address this complexity, the AutoML technique aims to build an automated pipeline for the full-process of machine learning (He et al., 2021). Neural Architecture Search (NAS), a significant sub-domain of AutoML, focuses on automatically discovering the neural architectures using machine learning or optimization methods. The NAS research encompasses three technical areas: the search space, the search strategy, and the evaluation strategy. Typically, search spaces are often designed hierarchically, comprising the macro architecture that determines the overall backbone of the network and the micro architectures that detail the inner-structure of each cell (block). Recent works have focused principally on four representative search spaces (Xie et al., 2021), including DARTS (Liu et al., 2019), NAS-RL (Zoph & Le, 2017), NAS-Net (Zoph, Vasudevan, Shlens, & Le, 2018), and MobileNet (Howard et al., 2017). Current popular search strategy spanning reinforcement learning (RL) (Lyu, Wen, Shi, & Huang, 2021; Zoph & Le, 2017; Zoph et al., 2018), differentiable architecture search, evolutionary search (EA) (Floreano, Dürr, & Mattiussi, 2008; Sun, Xue, Zhang, & Yen, 2019), and bayesian optimization (BO) (Zhou, Yang, Wang, & Pan, 2019). Some works Loni, Mousavi, Riazati, Daneshtalab, and Sjödin (2022), Loni, Zoljodi, et al. (2022), Mousavi, Loni, Alibeigi, and Daneshtalab (2023) present novel methods for optimizing neural network architectures, specifically focusing on sparsity and quantization to enhance inference efficiency in resource-constrained deployment situation.

PIM-based Software-hardware co-exploration. Instead of directly mapping the promising networks on CIM platforms, a number of recent methods have been developed to co-explore the optimal neural network model and CIM-based deployment scheme to jointly optimize performance and efficiency across multiple objectives, from accuracy and latency to energy and area (Jiang, Sha, Zhuge, Yang, Dong, & Chen, 2021). Under the device-circuit-architecture co-exploration framework, NACIM (Jiang et al., 2020) utilizes the hardware-aware NAS method to simultaneously determine the most promising neural architectures for the CIM accelerator. In NASCIM, the hardware space includes the device type and circuit topology, while its neural architecture space covers both the architecture hyperparameters and quantization scheme. In the pursuit of optimizing both accuracy and energy efficiency, Gibbon (Sun et al., 2022) employs an evolutionary algorithm-based search strategy for the co-design of the neural network model and PIM architecture. This approach specifically utilizes an RNN-based performance model – encompassing both accuracy and hardware efficiency – to significantly reduce the evaluation overhead. The focus of these related works is on hardware-aware NAS exploration on non-von-Neumann platforms, which involves the formulation of search space (architecture space, hardware device space) and the customization of search objectives (accuracy, latency, energy, etc.). Typically, heuristic search algorithms (such as RL-based or EA-based) are employed to explore the search space, guided by resource-aware evaluation metrics. BST-NAS (Lyu, Wen, et al., 2023) introduces a NAS methodology specifically aimed at investigating the optimal bit-level sparsity-tolerant architectures tailored for memristive platforms. Comparably, our work does not cover the entire pipeline of PIM/CIM hardware-aware NAS, it significantly contributes to the deployment scheme exploration for the full search space architecture. Specifically, we attempt not only to provide a full-space compatible inference deployment solution on memristive parallel analog platforms but also to serve as a part of the hardware-aware NAS, supporting the candidate architecture deployment to obtain resource-aware evaluation metrics.

3. Preliminary

3.1. Matrix–vector multiplication with memristor crossbars

The matrix–vector multiplication $W(m \times n) * v(n \times 1)$ can be efficiently computed by the memristor crossbar, with the synaptic weights being programmed into the memristors, which are represented by the conductance value, and the input vector represented as the voltage value, as shown in Fig. 1. The conductance value of the memristor must be positive in physics, whereas the weights matrix contains both positive and negative elements. To address this issue, this problem can be resolved by separating the weights matrix into two groups of synaptic weights matrices (Hu et al., 2012), where one matrix W^+ contains all the positive elements, as:

$$W^+_{(i,j)} = \begin{cases} W_{i,j} & W_{i,j} > 0 \\ 0 & else \end{cases} \tag{1}$$

the other matrix W^- contains all the negative elements:

$$W^-_{(i,j)} = \begin{cases} \left|W_{i,j}\right| & W_{i,j} < 0 \\ 0 & else \end{cases} \tag{2}$$

Thus the matrix W is separated as:

$$W = W^+ - W^- \tag{3}$$

the matrix–vector multiplication is denoted as:

$$W * v = (W^+ - W^-) * v = \begin{cases} W^+ * v + W^- * (-v) \\ W^+ * v - W^- * v \end{cases} \tag{4}$$

where the two results correspond to different crossbar circuit design methods. In the first computation pattern, the input vector v needs to be split into v and $-v$, and fed into the W^+ and W^-, respectively. This approach utilizes Kirchhoff's current law (KCL) to naturally facilitate the current accumulation. Conversely, for the second type, the input vector v is fed into two crossbars (W^+ and W^-), followed by a subtractor to complete the operation. As shown in Fig. 1, this figure illustrates the two kinds of computation patterns on the left and right sides, respectively. During the programming stage, the synaptic weights values are linearly transformed to the conductance domain in the range of $[\sigma_{min}, \sigma_{max}]$. The transformation functions are as follows:

$$\sigma^+ = \frac{(\sigma_{max} - \sigma_{min})}{max(\left|W_{i,j}\right|)} W^+ + \sigma_{min} \tag{5}$$

$$\sigma^- = \frac{(\sigma_{max} - \sigma_{min})}{max(\left|W_{i,j}\right|)} W^- + \sigma_{min} \tag{6}$$

For computation, each column j of the crossbar accomplish the computation by Eq. (7):

$$y_j^+ = -R_m \left(\sum_{i=1}^{n} \left[x_i \sigma^+_{i,j} - x_i \sigma^-_{i,j} \right] + x_b \sigma_b \right) \tag{7}$$

where R_m is the resistance of the programmable gain memristor in the op-amp circuit, denoted as:

$$R_m = \frac{max(\left|W_{i,j}\right|)}{(\sigma_{max} - \sigma_{min})} \tag{8}$$

For the first type of computing pattern, another amplifier should serve as the inverse transform circuit to reach the reverse current value that are to be fed into the next crossbar, as Eq. (9):

$$y_j^- = -\frac{R_f}{R_f} y_j^+ = -y_j^+ \tag{9}$$

This work employs the memristive multiplication design that consists of two separable crossbars and one subtractor, illustrated as the right part of Fig. 1. For the sake of conciseness, we do not detail the circuit design of our programming circuit design that is based on the method proposed in Yakopcic et al. (2016, 2017), in which two D-to-A converters (DACs) are utilized to implement a bounded voltage range for the programming of the synaptic weights. To maintain focus on our primary method, all programming circuits are omitted in this paper.

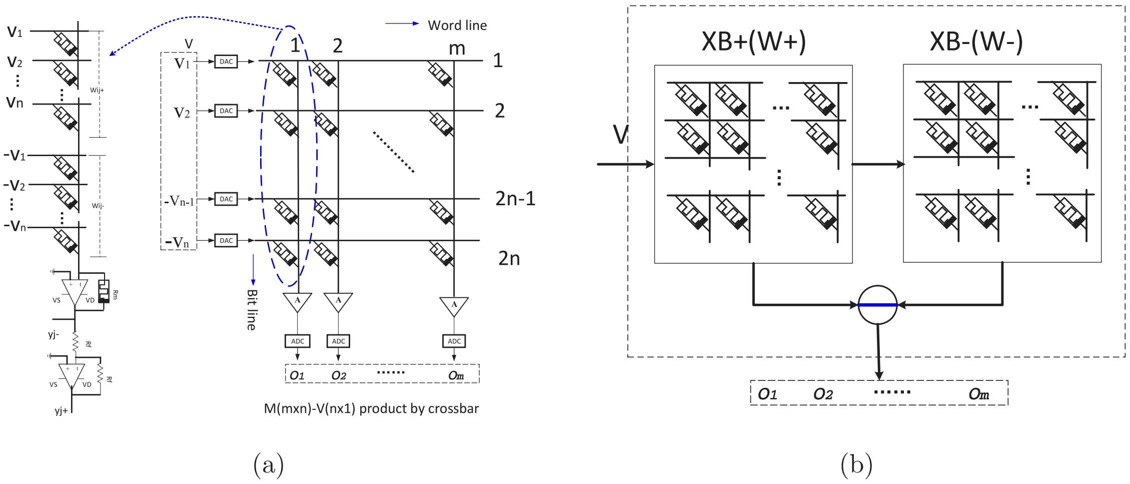


Fig. 1. Matrix–vector multiplication implemented by the memristor crossbars, in which Kirchhoff's current law (KCL) and Ohm's law realize the computation. (a) The input vector v needs to be formed as v and $-v$, and fed into the W^+ and W^-, respectively. (b) The input vector v is fed into two crossbars (W^+ and W^-), followed by a subtractor to complete the operation.

3.2. Convolution with memristor crossbars

The convolution operation with input i of dimension $\left(C_{\text{in}}, H_{in}, W_{in}\right)$, kernel of dimension w (k, k, C_{in}, C_{out}), bias vector b, and output o of dimension $\left(C_{\text{out}}, H_{\text{out}}, W_{\text{out}}\right)$ is formulated as:

$$\mathrm{o}\left(C_{\text{out}\ j}\right)=\sum_{k=0}^{C_{\text{in}}-1}\mathrm{w}\left(C_{\text{out}\ j},k\right)\star\mathrm{i}(k)+\mathrm{b}\left(C_{\text{out}j}\right) \tag{10}$$

where operation $\star$ is the 2D cross-correlation operator, C_{in} and C_{out} denotes input channel number and output channel number, respectively, $\mathrm{o}\left(C_{\text{out}\ j}\right)$ is one channel of the output feature map.

Whether based on traditional computing architecture (CPUs, GPUs), emerging computing architecture (TPUs, NPUs), or heterogeneous convergence platforms (ASICs), convolution operations are implemented by matrix–matrix multiplication or matrix–vector multiplication, so that the general acceleration libraries (*e.g.*, CuSParse, or CuDNN) are employed to achieve the acceleration. As shown in Fig. 2, there are usually two types of computing patterns for the convolution operation, one is the traditional way, which arranges the kernel elements in a redundant matrix, and the other one is the way adopted by Caffe (Jia et al., 2014), which schedule the elements of the feature maps in the matrix. Memristor crossbar-based convolution usually adopts traditional calculation methods due to the characteristics of parallel and simulation operation, as proposed by Shafiee et al. (2016) and Yakopcic et al. (2016), in which the kernel elements are permuted as a redundant matrix to meet the calculation law of convolution operation. For comparison, we also illustrate the arrangement of synaptic weights of single-channel convolution operator with different kernel sizes and strides in Fig. 2. For clarity, we omit the convolution modes, *i.e.*, full, valid, and same, as well as the padding operation.

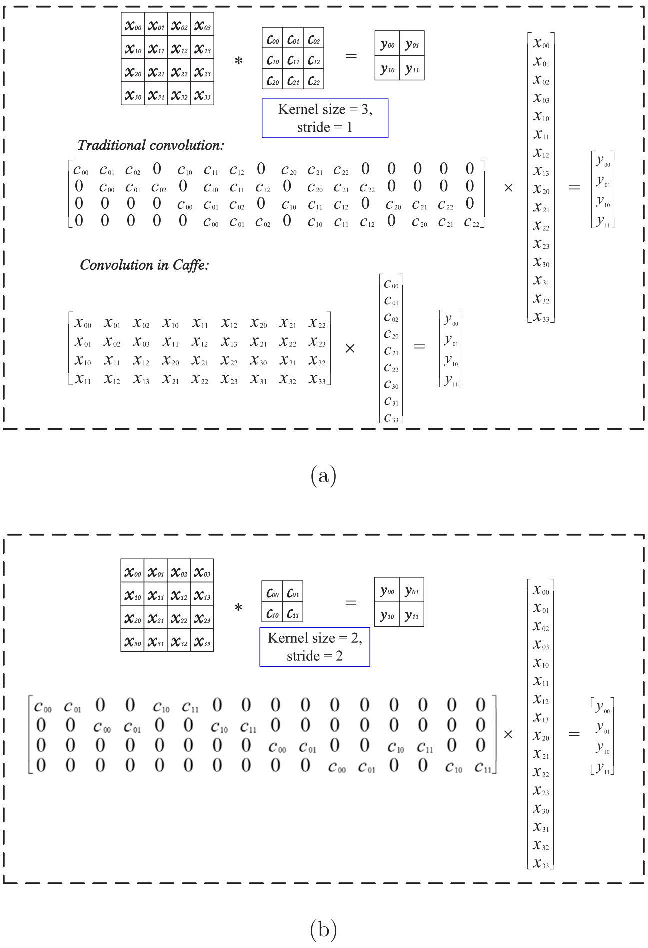


Fig. 2. The single-channel convolution operation that is implemented in the form of matrix–vector multiplication. (a) The kernel size of 3 × 3 with stride 1, presented by two types of M-V multiplication (traditional pattern and that in Caffe). (b) The kernel size of 2 × 2 with stride 2.

3.3. Neural architecture search and search space

Inspired by artificial neural networks, the current NAS search space covers 3 types of network backbones, including chain-structured, multi-branch, and cell-stacked (Elsken, Metzen, & Hutter, 2019). Based on these backbone designs, recent NAS research also concentrates on 4 representative search spaces, and the most popular ones are *MobileNet*-like and *DARTS*. This work targets the *DARTS* search space, which is cell-stacked and has good performance, limited scale of space, and well transferability across the tasks (Xie et al., 2021).

In *DARTS*, to make the search space continuous, the mixed-edge operation o in specified location (i,j) is characterized by all candidate operations in primitive operation set $\mathcal{O}$, as Eq. (11):

$$\bar{o}^{(i,j)}(x)=\sum_{o\in\mathcal{O}}\frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o'\in\mathcal{O}}\exp\left(\alpha_{o'}^{(i,j)}\right)}o(x) \tag{11}$$

by which the mixed-edge operation are parameterized by a vector $\alpha^{(i,j)}$ with dimension $|\mathcal{O}|$. As shown in Fig. 3, the fully connected DAG (cell) consists of a series of mixed-edge operations, and the weights-sharing

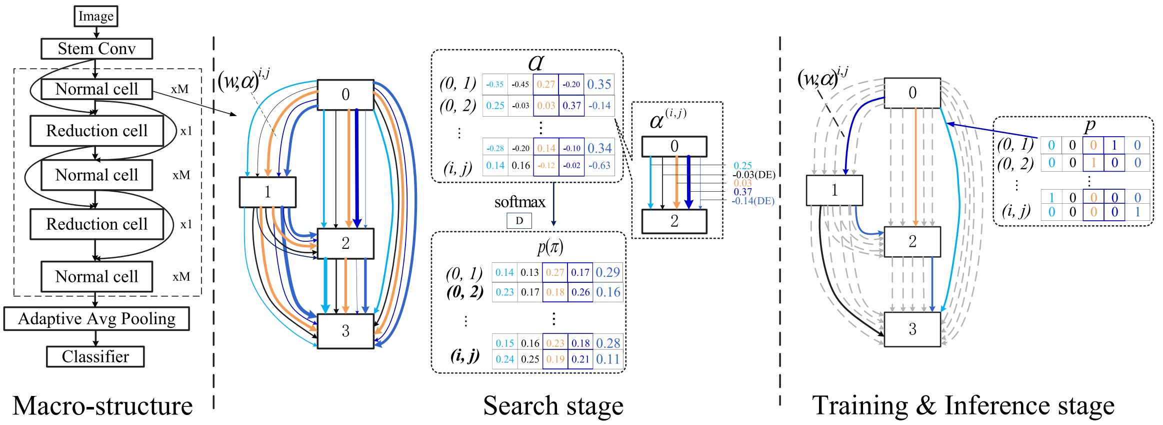
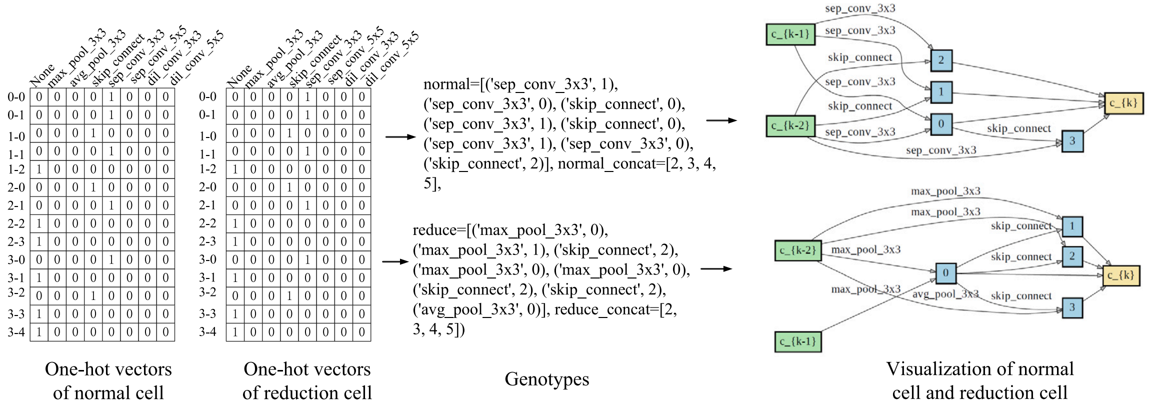


Fig. 3. Illustration of differentiable NAS during search, training, and inference. Left: Macro-structure of the hypernetwork during the search, and the candidate sub-network after the search, Middle: The Directed Acyclic Graph (DAG) cell that is constructed by the mixed-edge operations, each edge is parameterized by (w, α), Right: After the convergence of (w, α) in search procedure, one candidate architecture is emitted for the full training and deployment (inference) on the target device.



	None	max_pool_3x3	avg_pool_3x3	skip_connect	sep_conv_3x3	sep_conv_5x5	dil_conv_3x3	dil_conv_5x5
0-0	0	0	0	0	1	0	0	0
0-1	0	0	0	0	1	0	0	0
1-0	0	0	0	1	0	0	0	0
1-1	0	0	0	0	1	0	0	0
1-2	1	0	0	0	0	0	0	0
2-0	0	0	0	1	0	0	0	0
2-1	0	0	0	0	1	0	0	0
2-2	1	0	0	0	0	0	0	0
2-3	1	0	0	0	0	0	0	0
3-0	0	0	0	0	1	0	0	0
3-1	1	0	0	0	0	0	0	0
3-2	0	0	0	1	0	0	0	0
3-3	1	0	0	0	0	0	0	0
3-4	1	0	0	0	0	0	0	0

One-hot vectors
of normal cell

	None	max_pool_3x3	avg_pool_3x3	skip_connect	sep_conv_3x3	sep_conv_5x5	dil_conv_3x3	dil_conv_5x5
0-0	0	0	0	0	1	0	0	0
0-1	0	0	0	0	1	0	0	0
1-0	0	0	0	1	0	0	0	0
1-1	0	0	0	0	1	0	0	0
1-2	1	0	0	0	0	0	0	0
2-0	0	0	0	1	0	0	0	0
2-1	0	0	0	0	1	0	0	0
2-2	1	0	0	0	0	0	0	0
2-3	1	0	0	0	0	0	0	0
3-0	0	0	0	0	1	0	0	0
3-1	1	0	0	0	0	0	0	0
3-2	0	0	0	1	0	0	0	0
3-3	1	0	0	0	0	0	0	0
3-4	1	0	0	0	0	0	0	0

One-hot vectors
of reduction cell

normal=[('sep_conv_3x3', 1), ('sep_conv_3x3', 0), ('skip_connect', 0), ('sep_conv_3x3', 1), ('skip_connect', 0), ('sep_conv_3x3', 1), ('sep_conv_3x3', 0), ('skip_connect', 2)], normal_concat=[2, 3, 4, 5],

reduce=[('max_pool_3x3', 0), ('max_pool_3x3', 1), ('skip_connect', 2), ('max_pool_3x3', 0), ('max_pool_3x3', 0), ('skip_connect', 2), ('skip_connect', 2), ('avg_pool_3x3', 0)], reduce_concat=[2, 3, 4, 5])

Genotypes

Visualization of normal
cell and reduction cell

Fig. 4. Illustration of the representative architecture *DARTS-V1*. The genotypes of the `normal` cell and `reduction` cell are generated from the corresponding one-hot encoding vector set, and the architecture visualization is shown on the right side.

hypernetwork relies on the stack of the cells. Thus, the weight parameters w and architecture parameters α is to be co-optimized during the search procedure. The objective of NAS is to learn a set of continuous variables α, so it can be formulated as a bi-level optimization problem with α as the upper-level variable and w as the lower-level variable, as Eq. (12) shows.

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}\left(w^*(\alpha), \alpha\right) \\ \text{s.t.} \quad & w^*(\alpha) = \arg\min_{w} \mathcal{L}_{train}(w, \alpha) \end{aligned} \tag{12}$$

To effectively solve this bi-level optimization problem, *DARTS* approximates the solution by alternate optimization. The pipeline of differentiable NAS is shown in Fig. 3, including macro-structure, micro-structure, and the progressive search. The search space of the normal/reduction cell can be represented by the Directed Acyclic Graph (DAG). The representative architecture, *DARTS-V1*, shown in Fig. 4, a series of the one-hot vectors serves as the genotype of the architecture.

In terms of the scale of the search space, our approach adheres to the search space of vanilla-*DARTS*, wherein the primitive operations are listed as in Table 1. Assuming the number of primitive operations is 7+1 ("+1" refers to the zero operation that indicates the lack of connection) and the number of stages is 4, each relaxed cell (represented by a Directed Acyclic Graph or DAG) contains $2+3+4+5=14$ edges to be learned, leading to a total of 8^{14} possible configurations (architectures). Given that both normal and reduction cells are to be searched, the total number of architectures covered by this continuous search space is $8^{14} \times 8^{14} \approx 2 \times 10^{25}$ (Chen et al., 2019).

4. Methodology

We propose an innovative way to have the full search space architectures uniformly deploy on memristor hardware circuit without reprogramming. The method is to model primitive circuits for primitive candidate operations of the search space, and then build a memristive all-inclusive hypernetwork, by which the candidate architectures can be generated based on the genotypes (a set of one-hot encoding vectors). During the programming stage, the architecture encoding vectors are programmed to the hardware to formulate the sub-architecture of the hypernetwork, combined with the corresponding synaptic weights, so that the memristive all-inclusive hypernetwork is compatible with each candidate architecture in the search space. Therefore, the primary issues to be addressed in this work are:

(1) Memristive hardware circuit design of the primitive operations.
(2) Hardware architecture of the memristive all-inclusive hypernetwork, and its compatibility.
(3) Software-hardware co-design for resource-consumption saving.

4.1. Primitive operations modeling

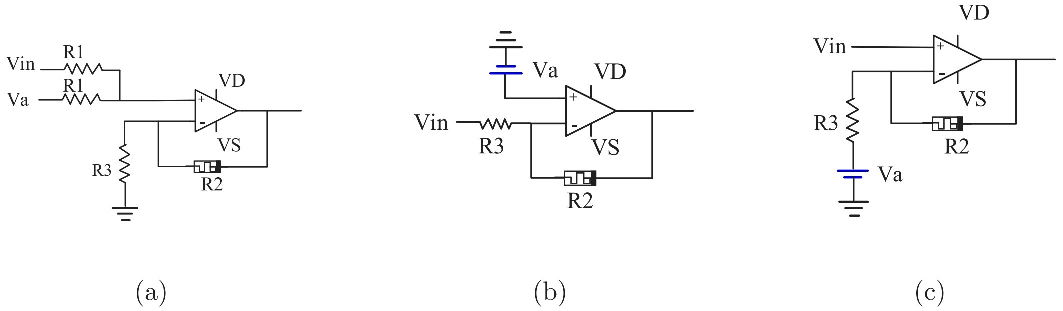
Our memristive hardware design starts from each primitive operation in Table 1, which is structured by a series of atomic operators.

4.1.1. Batch normalization

In neural networks, `batch normalization` (BN) (Ioffe & Szegedy, 2015) is widely used to alter the data distribution of the feature maps,

Table 1
Primitive operations in vanilla-DARTS search space.

Primitive operations	Structure
`none`	/
`skip_connect`	/
`max_pooling`	kernel size is 3
`avg_pooling`	kernel size is 3
`sep_conv_3` × 3	`ReLU` + `Depthwise_Conv` (kernel size is 3) + `Pointwise_Conv` + `BN`
`sep_conv_5` × 5	`ReLU` + `Depthwise_Conv` (kernel size is 5) + `Pointwise_Conv` + `BN`
`dil_conv_3` × 3	`ReLU` + `Depthwise_Conv` (kernel size is 3, with dilation 2) + `Pointwise_Conv` + `BN`
`dil_conv_5` × 5	`ReLU` + `Depthwise_Conv` (kernel size is 5, with dilation 2) + `Pointwise_Conv` + `BN`



(a) (b) (c)

Fig. 5. Three types of the circuit design of `batch normalization`, in which the negative feedback of the operational amplifier is used to perform linear mapping multiplication. (a) Two resistors and a programmable voltage source to construct the add item (b) in Eq. (20). (b) A programmable voltage source to construct the add item on the positive pole of the op-amp. (c) A programmable voltage source to construct the add item on the negative pole of the op-amp.

aiming to facilitate training. During the inference stage, the `BN` process involves two primary steps: first, it normalizes the data distribution x to x' using the statistical mean value and variance, and then scales and shifts x' to a new distribution y with a scale factor γ and a shift factor β. The formula for the `BN` layer is calculated as:

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}} * \gamma + \beta \tag{13}$$

where $\mathrm{Var}[x]$, $E[x]$, is the mean value and variance, respectively, which are statistically calculated during the training stage. Specifically, for each mini-batch in training:

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \tag{14}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^{m} \left(x_i - \mu_{\mathcal{B}}\right)^2 \tag{15}$$

and to achieve the statistical global data distribution, mean and variance are calculated by the running mean:

$$\mu_{\text{statistic}+1} = (1-m) * \mu_{\text{statistic}} + m * \mu_{\mathcal{B}} \tag{16}$$

$$\sigma_{\text{statistic}+1}^2 = (1-m) * \sigma_{\text{statistic}}^2 + m * \sigma_{\mathcal{B}}^2 \tag{17}$$

where m is the momentum parameter, which is commonly set as 0.1. The final values of $\mu_{\text{statistic}}$ and $\sigma_{\text{statistic}}^2$ act as the $\mathrm{E}[x]$ and $\mathrm{Var}[x]$ of the dataset, respectively. These values become fixed for each layer in the ex-situ memristive network implementation, ensuring consistency across the network. In terms of Eq. (13), linear transform can be simplified as:

$$\begin{cases} y = k \cdot x + b \\ k = \dfrac{\gamma}{\sqrt{\mathrm{Var}[x]+\varepsilon}} \\ b = \beta - \dfrac{\gamma \cdot \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\varepsilon}} \end{cases} \tag{18}$$

where k, b are determined by pre-computing, then the multiplication and addition can be conducted by the memristive hardware circuits.

We propose three types of circuit design schemes to realize `batch normalization` in the analog domain with an op-amp, as shown in Fig. 5. In terms of Fig. 5.(a), the computation law of circuit follows Eq. (19):

$$V_o = \frac{1}{2}\left(1 + \frac{R_2}{R_3}\right)\left(V_{IN} + V_A\right) = \left(\frac{1}{2} + \frac{R_2}{2R_3}\right)\left(V_{IN} + V_A\right) \tag{19}$$

thus following Eq. (20):

$$k = \frac{1}{2} + \frac{R_2}{2R_3}, b = \left(\frac{1}{2} + \frac{R_2}{2R_3}\right) \cdot V_A \tag{20}$$

we may adjust the R_2, R_3 resistors according to a pre-determined k value to implement each batch normalization layer. R_1 is a resistor with an arbitrary resistance value, and V_A is the input voltage that can be adjusted according to the pre-determined b value. The other two design schemes are based on pre-defined voltage values during the programming. In our simulation, we employ the first scheme.

4.1.2. Normal convolution

As proposed in the preliminary section, the primitive `normal convolution` operators in NAS are represented by a batch operator `ReLU+Conv+BN`, in which rectifier linear unit (`ReLU`) is implemented by the diode, to realize the: $ReLU(x) = max(0, x)$, hardware circuit is shown in Fig. 6. We follow the second convolution pattern described in Fig. 1, which separates the W^+ and W^- crossbar and is followed by a subtractor to accomplish the calculation.

4.1.3. Max pooling

Our previous work proposes a `max pooling` circuit based on the maximum voltage selector circuit (Wen, Wei, et al., 2019; Wen, Wei, Zeng, & Huang, 2018), in which two NMOS transistors (MN1 and MN2) are connected to select the maximum value of $V1$ and $V2$, as shown in Fig. 7. Thus the `max pooling` primitive operation with an input size of $(n \times n)$ requires $(n^2 - 1)$ maximum voltage selectors.

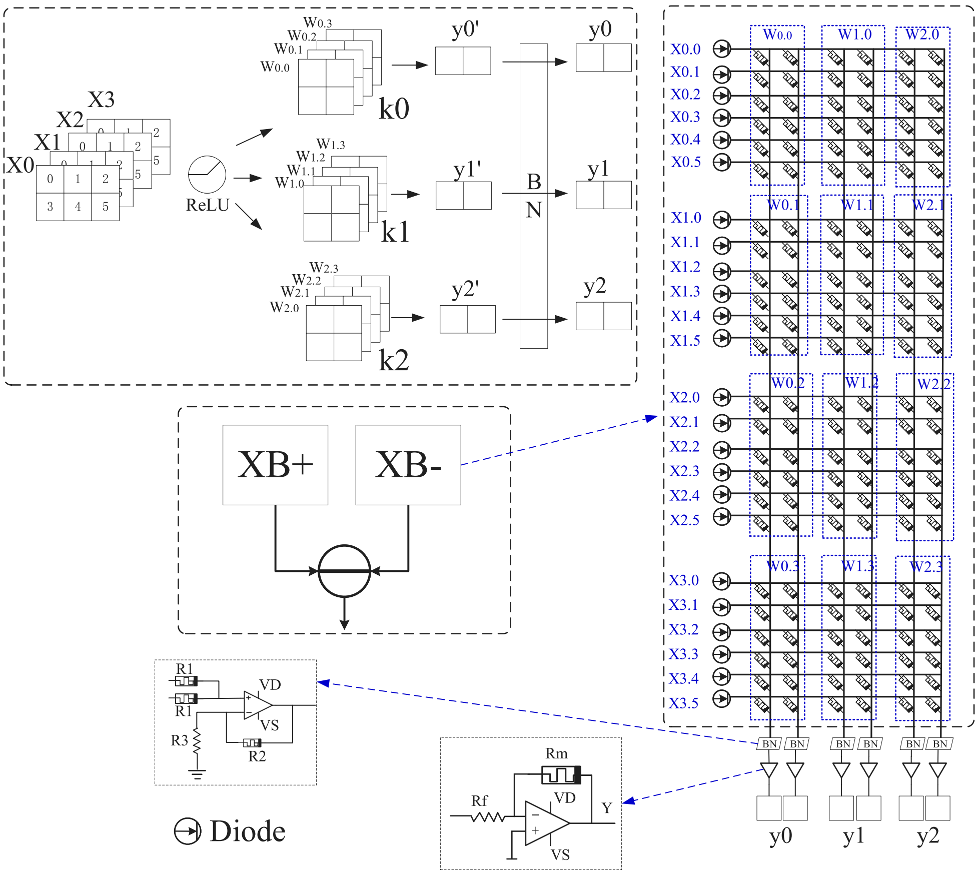


Fig. 6. The circuit design of the normal convolution operator in NAS. It consists of the ReLU (implemented by diode), convolution, and batch normalization.

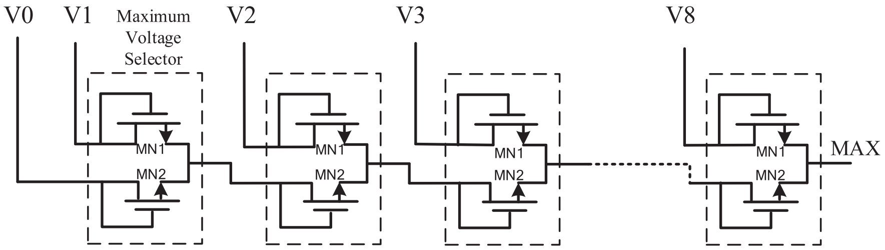


Fig. 7. The circuit design of the max pooling with input size 3 × 3.

4.1.4. *Average pooling*

An 8 × 8 single-channel average pooling operation can be effectively realized by 64 memristors, each having a conductance value of 1/64. Fig. 8 illustrates the circuit design. In this design, each column of the memristor crossbar circuit corresponds to a calculation of the average pool in one channel of the feature maps, which is then followed by an op-amp circuit that performs gain amplification to obtain the output value.

4.1.5. *Separable convolution*

The Separable convolution (Howard et al., 2017) is composed of the ReLU, depthwise convolution, pointwise convolution, and batch normalization. In the depthwise convolution, convolution is performed on each channel of the feature maps using a convolution kernel that has only one channel. For the pointwise convolution, a simple 1 × 1 normal convolution is utilized, ensuring the feature maps maintain the spatial resolution, while the channel number of output feature maps matches the number of 1 × 1 filters. The circuit design scheme of the primitive separable convolution is shown in Fig. 9, where the memristor conductance values are marked in different colors to distinguish the arrangement of crossbars.

4.1.6. *Dilated convolution*

The dilated convolution is a standard convolution operation with space interval sampling to increase the reception field. Compared with normal convolution, dilated convolution has one additional hyperparameter–dilation rate, which refers to the spatial interval pixels (*e.g.*, normal convolution's dilation rate is 1), as illustrated in Fig. 10. For conciseness, we do not present the concrete circuit design of dilated convolution for its similarity with normal convolution.

4.2. *Memristive all-inclusive hypernetwork*

To ensure the adaptation of the feature maps that are generated by alternative candidate normal cell and reduction cell architectures (micro), we propose the memristive all-inclusive hypernetwork, in which the channel number and the spatial size of the feature maps (FMs) should be elaborately designed.

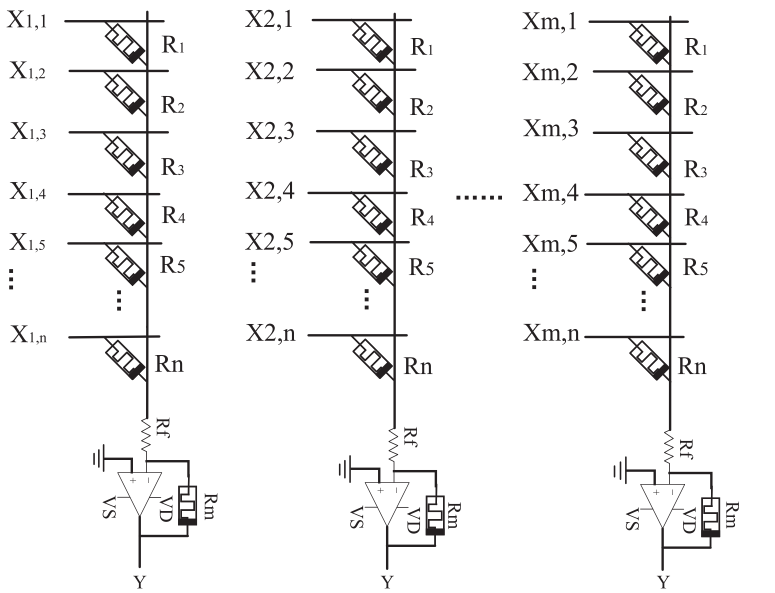


Fig. 8. The circuit design of the `average pooling`, in which each column of the crossbar represents one sliding window of the feature maps with n elements.

Table 2
The exemplified cell-level adaptation configuration (initial channel number is 12, the stage of one cell is 4, and the cell number is 8).

Cells	Pre_Pre _Channel	Pre_ Channel	Primitive_ Operation_ Channel	Output_ Channel	Stride	Spatial_ Size
Stem	–	3	–	**36 (3C)**	1	**32 (H0)**
Normal	36	36	12 (C)	48 (4C)	1	32
Normal	36	48	12	48	1	32
Reduction	48	48	24 (2C)	96	4	8 (H1: H0/Stride)
Normal	48	96	24	96	1	8
Normal	96	96	24	96	1	8
Reduction	96	96	48 (4C)	192	4	2 (H2: H1/Stride)
Normal	96	192	48	192	1	2
Normal	192	192	48	192	1	2
Adaptive_ Avg_Pooling	–	–	–	–	–	**1**
Classifier	192	–	–	–	–	10

4.2.1. *Spatial adaptation of FMs*

(1) **Adaptation in primitive operations level**: The input feature maps ($K-1$ and $K-2$) are pre-processed to change the resolution (in reduction cell) for adaptation. All the primitive operations do not change the resolution, except for the first 2 primitive operations in the reduction cell (by $stride = 2$).

(2) **Adaptation in cell level**: The `normal cell` keeps the spatial resolution, and the `reduction cell` reduces the spatial resolution by a factor of 4. The final adaptation average pooling layer reduces the resolution to 1.

4.2.2. *Channel adaptation of FMs*

Our design for compatibility is guaranteed by the adaptation design at the primitive operation level and cell level:

(1) **Adaptation in primitive operation level**: The candidate primitive operations are designed to maintain the channel number, but during the concatenation in each cell, this process extends the channel number from C to 4C, *e.g.*, achieved by `the concatenation of the FMs` with indices $[2,3,4,5]$.

(2) **Adaptation in cell level**: The `normal cell` is configured to maintain the channel number, while the `reduction cell` is designed to increase the channel number.

Taking the initial channel number as 12, the stage of one cell as 4, and the cell number as 8 for instance, the configuration of the entire network is shown in Table 2. This setup includes channel expansion and spatial resolution reduction, which are achieved in the `Reduction cell` (marked brown) and the first `Normal cell` (marked blue), respectively. The fixed part of the hypernetwork is the first preprocessing layer (`Stem`) which prepares the data and the last adaptive `average pooling` layer for final output processing. Therefore, from the perspective of Feature Maps (FMs) in the hypernetwork, to achieve compatibility with different candidate architectures, the width (the spatial resolution $H \times W$), the depth (cell number), and the thickness (channel number) are maintained consistently. That is, the dimensions and data fusion of all the FMs of all candidate architectures included by the all-inclusive hypernetwork remain consistent, while the candidate primitive operations vary, offering a compatible and scalable approach in the search space, as Fig. 11 shows. During the programming stage, regarding one normal cell or reduction cell, the one-hot architecture vector representing the genotype controls the output gates of the mixed-edges to dynamically select the primitive operation, as demonstrated in the control flow diagrams. Importantly, without reprogramming the synaptic weights, the circuit of the atomic operator (the bottom level) with different kernel sizes and strides cannot be uniformly designed.

4.3. *Fixed-point quantization neural network for low-bit ADCs*

The current study statistically shows that analog-to-digital converters (ADCs) account for most of the resource consumption in PIM

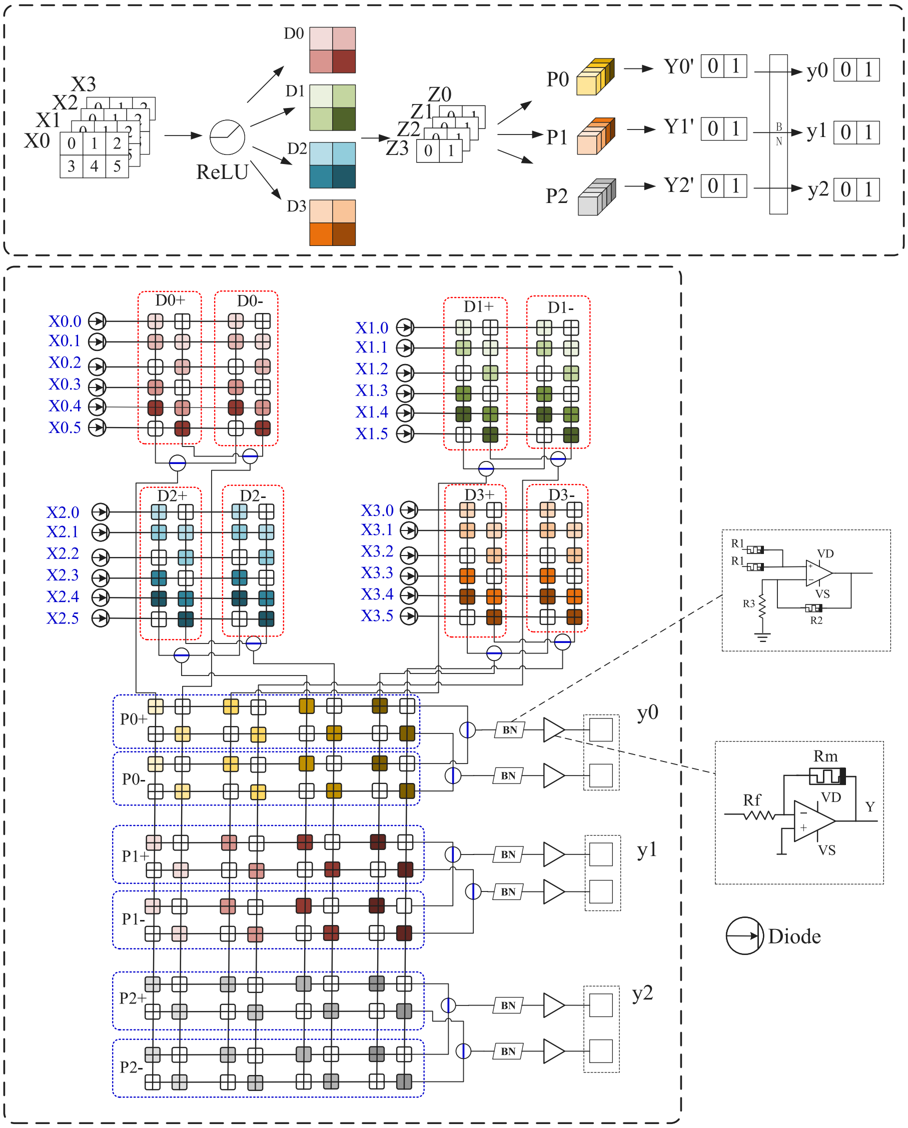


Fig. 9. Schematic representation of the `separable convolution` operator in NAS, encompassing ReLU, `depthwise convolution`, `pointwise convolution`, and `batch normalization`. An op-amp is utilized to convert the current to the output value. In this design, following the depthwise crossbar, the pointwise crossbar integrates the information of the feature maps while preserving their shape.

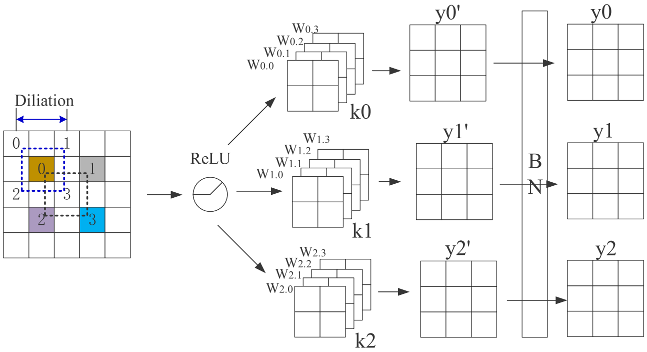


Fig. 10. The schematics illustrating the circuit design of the `dilated convolution` operator in NAS. This design maintains consistency with `normal convolution`, but incorporates a spatial interval to enlarge the receptive field. Distinct from `normal convolution`, `dilated convolution` introduces an additional hyperparameter called dilation rate, which defines the number of kernel intervals (*i.e.*, `normal convolution` is equivalent to dilated convolution with a dilation rate of 1).

platforms (58% in energy consumption and 81% in area consumption (Huang et al., 2021)). Fixed-point quantization can effectively reduce the dynamic range of network activation values at each layer so that even with low-bit ADC sampling precision, it can meet the inference accuracy requirements (Zhang, Yang, Chen, Wang, & Li, 2019). Since the positive and negative elements of the synaptic weights matrix will be separated into two different crossbars, we will only discuss the positive weights. The dynamic range of a layer W_l is defined as Eq. (21):

$$S\left(W_l\right) = \left\lceil \log_2 \left(\max_{w_l^i \in W_l} \left(\left| w_l^i \right| \right) \right) \right\rceil \tag{21}$$

where i is the weight index and l is the layer index, and, $S\left(W_l\right)$ is the scale factor. Considering the n-bit precision, the quantization step size would be:

$$Q_{\text{step}} = 2^{S(W_l)-n} \tag{22}$$

the quantization operation of weight w_l^i follows Eq. (23):

$$B\left(w_l^i\right) = \left\lfloor \frac{w_l^i}{Q_{\text{step}}} \right\rceil . \tag{23}$$

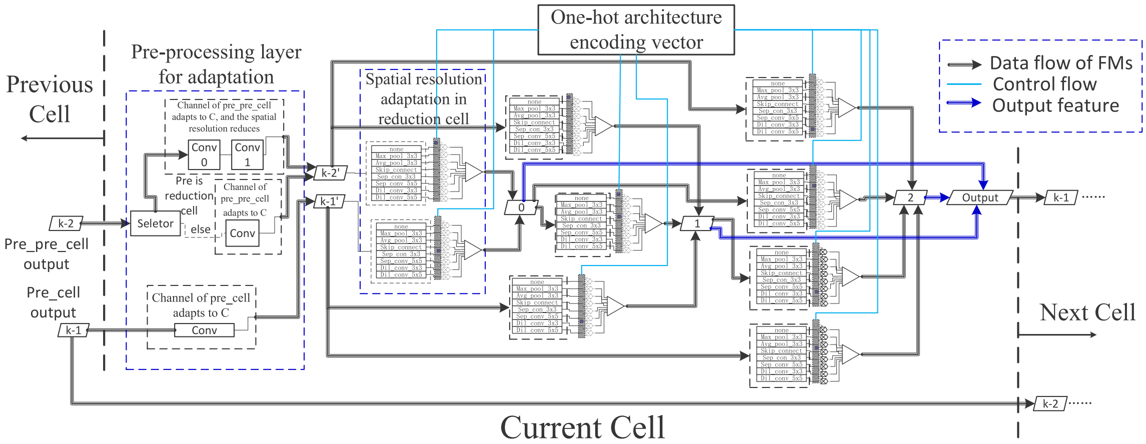


Fig. 11. Circuit design schematics for a NAS-based cell, forming part of the memristive all-inclusive hypernetwork. This illustration simplifies the design by showing a cell with 3 stages, where the number of feature maps is 3, and the total number of mixed-edges in the fully-connected Directed Acyclic Graph (DAG) is $2+3+4=9$. A one-hot architecture vector is used to control the output gates of the mixed-edges, enabling dynamic selection of the primitive operation to accommodate the full search space. Additionally, the output feature maps of the current cell become the input feature maps $(k-1)$ for the next cell.

which maps all $B\left(w_l^i\right)$ to range $[0, 2^n-1]$. The n-bit binary weights value $B\left(w_l^i\right)$ will be programmed into the memristors. In the forward process during training, the recovered quantization value is calculated as Eq. (24):

$$q^{(t)} = Q\left(w_l^{(t)}\right) = B(w_l^t) \times Q_{step} \tag{24}$$

For the backward propagation, the gradient of cross-entropy loss is applied on $q^{(t)}$ with the full precision, as Eq. (25):

$$w_l^{(t+1)} = q^{(t)} - lr_w \times \nabla_q \mathcal{L}_{train}\left(q^{(t)}\right) \tag{25}$$

After ex-situ training, the n-bit binary fixed-point weights $B(w_l^t)$ are programmed into the memristor crossbar to conduct the inference.

5. Experiment

5.1. Memristor model

This work utilizes the threshold memristor model (Jo et al., 2010), with distinct threshold characteristics, and the resistance characteristic is represented by:

$$i(t) = G(t)v(t) \tag{26}$$

where $v(t)$ is the voltage, $i(t)$ is the current, and $G(t)$ is the conductance value. The resistance of memristor changes when the applied current or the voltage exceeds the threshold, according to the law of the differential equation:

$$\frac{dG}{dt} = T(v(t))W(G(t), v(t)) \tag{27}$$

where $W(R(t), v(t))$ is a window function that is used to limit the resistance of the threshold memristor between *Roff* and *Ron*, is defined as follows:

$$\begin{aligned} W(G(t), v(t)) &= \theta(-v(t))\theta(G(t) - G0) \\ &+\theta(v(t))\theta(G1 - G(t)) \end{aligned} \tag{28}$$

where $\theta()$ is the sign function, by which if the argument is positive, it returns 1, otherwise 0. The conductance value is limited between G_0 and G_1. With the window function, the change rate of the conductance value of the memristor becomes 0 when the conductance reaches the upper or lower bound. $T(v(t))$ is a voltage threshold function:

$$T(v(t)) = \beta\left[v(t) - 0.5\left(\left|v(t) + V_t\right| - \left|v(t) - V_t\right|\right)\right] \tag{29}$$

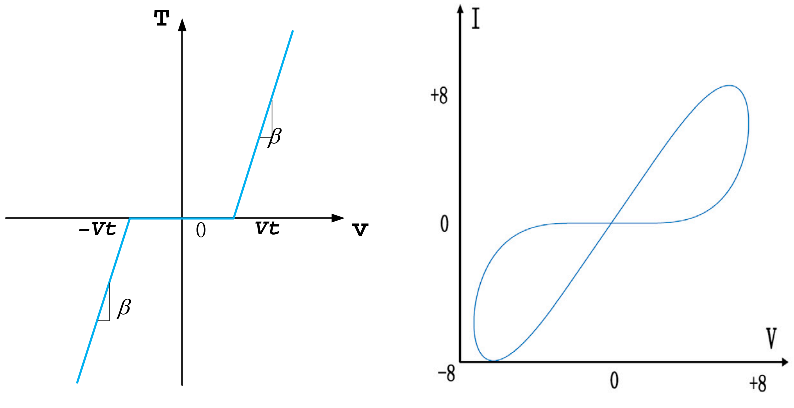
As Fig. 12 shows, when $v(t)$ is within the range of $[-v(t), v(t)]$, $T(v(t))$ remains zero. In contrast, outside this range, $T(v(t))$ changes with the gradient β, which controls the rate of change. Assuming β is sufficiently large, once the applied voltage exceeds the threshold, the conductance will rapidly fluctuate, switching between $[G_0, G_1]$. Conversely, when β is small, the conductance of the memristor undergoes a gradual change once the voltage exceeds the threshold. Following the methodology of previous works (Wen, Wei, et al., 2019; Wen et al., 2021), we set β to be 1, and ensure that the voltage is kept lower than the threshold V_t during the inference stage, and greater than the threshold in the programming stage.

5.2. Experiment and analysis of computational cost

Our ex-situ training experiments were conducted using the PyTorch 1.4 framework with CUDA 9.0 on 2 NVIDIA $1080TI$ GPUs, each with $11GB$ of GPU memory. The memristor hardware simulation experiment is based on the Simulink toolbox. We chose several representative outstanding network architectures (*DARTS-V1*, *DARTS-V2*, *PDARTS*) from the *DARTS* search space to perform the ex-situ training and simulation. Table 3 shows the comparison among these outstanding *DARTS*-like networks and hand-tuned artificial networks in terms of performance, efficiency, search time cost, and GPU-memory consumption. This comparison reveals that NAS-based architecture can achieve better efficiency (*e.g.*, Parameters) and performance.

Due to the limitation of circuit simulation difficulty, we reduce the scale of the memristive all-inclusive hypernetwork, we set the primary hyperparameters *Initial channel numbers* to be 8/12, and *Layers number* to be 8/12, which determines the scale of the model, respectively. In addition, since the structural limitation of the outstanding architectures (*DARTS-V1, DARTS-V2, PDARTS*), the *Stage number* is fixed to be 4. The detailed configuration of the memristive all-inclusive hyper-network is shown in Table 2, which shows the change in the feature map channel number and the spatial resolution. Our training details follow the experimental setting of evaluation in *PDARTS* (Chen et al., 2019), detailed hyperparameters are shown in Table 4. In terms of the hardware circuit simulation, our experiments are conducted based on a subset of the test set in CIFAR10.

To explore the influence of quantization precision on inference accuracy, we have undertaken 16/8/4-bit fixed-point quantization training, respectively. The final experimental results are quantitatively shown in Table 5. As observed, with the decrease in quantization precision, the performance of the network tends to deteriorate; thus, intuitively, a balance must be struck between performance and resource consumption. Significantly, our primary focus lies in the implementation of the memristive circuit and its simulation to verify its effectiveness, as opposed



(a)   (b)

Fig. 12. (a) The voltage threshold function and (b) The V-A curve of threshold memristor.

Table 3
Ex-situ training and simulation results.

Architecture		Test Error (%)	Params (.M)	Search Cost (.GPU-days)	Memory Cost (.GB)
Manual	DenseNet	4.1	7.0	-	-
	DenseNet-BC	3.46	25.6M	-	-
	ResNet	6.61	1.7	-	-
NAS	DARTS-V1	3.00 ± 0.14	**3.3**	1.5	**11**
	DARTS-V2	2.76 ± 0.09	**3.3**	4	11
	PDARTS	**2.50**	3.4	0.3	16
	PC-DARTS	2.57	3.6	**0.1**	12

Table 4
Hyperparameters of ex-situ training.

Batch size	512	Initial channels	8/12
Epochs	400	Layers	8/12
Optimizer	SGD	Weight decay	3e−4
Learning rate	0.025	LR decay	cosine
Momentum	0.9	Quantization	16/8/4 bit

to improving the accuracy of the classification task. Compared with the original full precision model, the quantization model exhibits degraded performance due to the loss of knowledge representation ability. Our computational simulation experiments were conducted only on 8-bit quantization precision, yielding a promising result of 91.6% on the *PDARTS* architecture using our memristive all-inclusive hypernetwork.

5.3. Hardware simulation and analysis

To measure the hardware-related metrics of the memristive all-inclusive hypernetwork, such as power, energy, and area, we conducted simulation comparison experiments with manually designed architectures, and individual NAS-based architectures. Some simulation tools have been proposed as PIM accelerator benchmark platforms (Peng, Huang, Jiang, Lu, & Yu, 2021; Xia et al., 2018), and our experiment was based on a memristor-based PIM simulation platform, MNSIM 2.0 (Zhu et al., 2020) (the 2nd version of MNSIM (Xia et al., 2018)), which simulates the behavior-level model for computing accuracy and hardware performance, including energy, power, area, etc. Based on GPU acceleration, MNSIM achieves more than 7000× speed-up compared with SPICE, but with 1% error bound relative to SPICE simulation results. Due to the limitations of MNSIM, we carried out the simulation without non-ideal factors, such as SAF, and resistance variations. Typical configuration parameters are shown in Table 6. For other configurations like ADC/DAC power, ADC/DAC area, and crossbar area, we adhered to the default configuration of the MNSIM platform. To adapt the NAS-based architecture, we referred to the *DARTS* computation graph and modified the MNSIM computation graph to support the irregular forward propagation process (DAG cell). The simulation results for hardware performance are detailed in Table 7. The individual NAS-based architectures (*DARTS-V1*, *DARTS-V2*) demonstrated significant superiority over manually designed networks in terms of hardware efficiency, namely in area, latency, and power. Generally, the NAS-based architectures feature fewer channel numbers and more complex and deeper connections, leading to a smaller model scale (`Parameters` and `FLOPS`). For DARTS architectures (*DARTS-V1*, *DARTS-V2*), additional consumption is observed when deployed on the memristive all-inclusive hypernetwork (*Hyper-DARTS-V1*, *Hyper-DARTS-V2*) relative to their individual deployment: First, the area of Hyper-DARTS increases significantly, approximately 1.5× relative to the original. This increase is inevitable, as the mixed-edge requires crossbar deployment for all candidate primitive operations (except for the `skip-connection`, and `pooling`), and the adaptation layers also contribute to the additional crossbar area. Second, the change in power is slight, while the inference latency experiences an increase by 22% ∼ 25%, resulting in additional overall energy consumption (increases by 40% ∼ 45%). This extra inference latency is attributed to the adaptive layers of intra- and inter-cell. Importantly, relative latency and cost are the most concerning issues, especially since our focus is on the memristive simulation performance comparison between the NAS-based architecture's sole deployment and its all-inclusive hypernetwork deployment. While our experiments are based on the *DARTS* full search space, in practice, the scale of the architecture space is scalable, which will also change the resource and energy efficiency, thus presenting a trade-off. To explore this, we empirically selected several groups of hyperparameter settings, and the detailed comparison results are shown in Table 8.

6. Conclusion

The current memristive hardware models target the implementation of manually designed network architectures. However, for the

Table 5
Ex-situ training and simulation results.

Architecture	Initial channel	Stages	Cells	Params (.M)	FLOPs (.M)	Quantization precision	Accuracy	
							Ex-situ training	Simulation
DARTS-V1	12	4	8	0.136	25.76	16	91.78	/
						8	91.64	87.05
						4	64.69	/
	8	4	12	0.098	20.098	8	85.72	81.6
	12	4	12	0.204	40.22	8	91.84	87.83
DARTS-V2	12	4	8	0.145	27.1	16	92.14	/
						8	91.93	88.3
						4	72.66	/
	8	4	12	0.104	21.31	8	87.46	84.7
	12	4	12	0.217	42.53	8	92.02	88.65
PDARTS	12	4	8	0.173	30.2	16	93.0	/
						8	92.9	89.2
						4	74.1	/
	8	4	12	0.119	22.9	8	86.15	83.0
	12	4	12	0.243	45.21	8	93.2	89.11

Table 6
Simulation typical configuration.

Simulation configuration	Level	Value/Choice
Weight bit width	Architecture	8
Activation bit width	Architecture	8
Crossbar size	Crossbar	256 × 256
ADC resolution	Interface	8
DAC resolution	Interface	2
Memristor model	Device	RRAM
Cell type	Device	0T1R
Device tech	Device	45 nm

current popular NAS-based model that outperforms the artificial convolution networks, due to the ever-varied and heterogeneous architecture pattern (*e.g.*, 10^{25}), the customized memristive circuit cannot uniformly support various models. Therefore, it has brought challenges to the processing-in-memory deployment of various task-specific models (*e.g.*, classification and segmentation tasks in computer vision).

Our work primarily addresses this issue and proposes a memristive all-inclusive deployment framework. We first model all primitive operations in *DARTS*, a leading search space in the NAS field. Then we propose the memristive all-inclusive hypernetwork that is adaptable to a 2×10^{25} size search space without further re-design, contrasting with previous works that were limited to fixed-structure neural network models. Experimentally, we conduct the ex-situ training on 3 outstanding architectures (*DARTS-V1*, *DARTS-V2*, *PDARTS*), where quantization training is utilized to reduce the model scale, and also optimizes ADCs in terms of energy and area consumption. Our simulation experiment with 8-bit quantization precision demonstrates promising results on memristor-based crossbars, achieving 89.2% of *PDARTS* with 8-bit quantization precision. The memristive all-inclusive hypernetwork (*Hyper-DARTS-V1*, *Hyper-DARTS-V2*) shows slightly higher resource consumption (power, energy, latency) compared to individual deployment, but remains feasible for industrial scenarios.

The memristive circuits and architectures have various synapse and neuron cell modeling types, depending on different memristive devices with diverse physical and material properties (Krestinskaya et al., 2019). While important, these factors are not the focus of this work, since our primary aim is the innovation of adaptation modeling of NAS-based architectures. While our experiments use the threshold memristor model, the proposed method is potentially generalizable to other analog PIM architectures. Undoubtedly, there exist some limitations in this work. First, factors such as SAF and resistance variations in the hardware performance simulation are not considered. Additionally, we only target the *DARTS* search space (backbone), since it has been proven to be efficient, performing exceptionally well performance on NAS tasks. Importantly, its design is relatively structured, simple, and represented by cells constructed from regular directed acyclic graphs. Based on this, we find it easier to utilize a somewhat redundant circuit design to construct a memristive all-inclusive hypernetwork for the compatible deployment requirement, which can also be composed of stacked cells (circuit). In contrast, the applicability analysis of our method in other spaces is as follows:

(1) **NAS-RL**: The NASNet space is not based on stacked cells but rather constitutes a cooperative search space with macro-architecture and micro-architecture spaces (DenseNet-like). Therefore, our method can be transferred to the micro-architecture part of this space, but not to the macro-architecture part.

(2) **MobileNet**: The MobileNet search space is constructed based on a purely chain-like backbone, formed by stacking MobileNet bottlenecks. The variations in architecture points depend on layer-by-layer changes in expansion ratio, stride, kernel size, etc., and do not originate from changes in primitive operations on specific feature map connections. Hence, it is challenging to construct a compatible deployment method between architecture points in this space. Currently, our design is unable to encompass such a search space, suggesting an avenue for future exploration and study.

CRediT authorship contribution statement

Bo Lyu: Conceptualization, Investigation, Methodology, Software, Writing – original draft, Writing – review & editing. **Yin Yang:** Formal analysis, Writing – review & editing. **Yuting Cao:** Investigation, Methodology. **Tuo Shi:** Writing – review & editing, Methodology. **Yiran Chen:** Methodology, Resources. **Tingwen Huang:** Formal analysis, Writing – original draft, Writing – review & editing. **Shiping Wen:** Supervision.

Declaration of competing interest

There is no conflict of interest in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This publication was supported by the National Key R&D Program of China under Grant No. 2021YFB3601200, the National Natural Sci-

Table 7
The Memristive simulation results of Hardware performance.

Metrics		Individual NAS architecture		Memristive all-inclusive hypernetwork	
		DARTS-V1	*DARTS-V2*	*Hyper-DARTS -V1*	*Hyper-DARTS -V2*
Area (um^2)	Entire	249936864.5	230205006.7	**374905296.6**	**361750724.8**
	Crossbar	229512314.9	211392921.6	344268472.3	332188876.8
	DAC	25837.57	23797.7	38756.4	37396.5
	ADC	8025600	7392000	12038400	11616000
	Buffer	7710257.0	7101552.5	11565385.5	11159582.5
Power (W)	Entire	11.9	10.9	**12.1**	**11.9**
	Crossbar	0.752	0.687	0.556	0.56
	DAC	0.430	0.395	0.623	0.61
	ADC	10.175	9.376	10.16	10.03
	Buffer	0.454	0.417	0.661	0.64
Energy (nJ)	Entire	1038883.3	935477.7	**1511572.6**	**1301511.5**
	Crossbar	21540.7	19319.2	25458.9	22873.6
	DAC	31056.1	27356.5	66801.1	56495.7
	ADC	965236.3	869981.1	1376290.9	1185640.7
	Buffer	20634.1	18438.9	42344.5	35909.6
Latency(ns)		6875989.9	6464115.3	**8408426.8**	**8112018.3**

[†]: The initial channel number is 12, the stage number of the DAG cell is 4, and the cell number is 8.

Table 8
The tradeoff between search space scale and (entire) hardware efficiency.

Primitive operation set scale	Stage number	Search space scale	*Hyper-DARTS-V1/DARTS-V1*			
			Area (um^2)	Power (W)	Energy (nJ)	Latency (ns)
8	4	2×10^{25}	374905296.7	12.1	1511572.6	8408426.8
5	4	3×10^{19}	322107913.8	11.82	1391152.6	7806571.8
5	3	9×10^{12}	291637424.5	11.9	1317233.3	7254586.2
3	2	6×10^{4}	225543840.5	11.76	1063465.0	6543561.1
1	4[a]	1[b]	249936864.5	11.93	1038883.3	6875989.9

[a] The *DARTS-V1* architecture's stage number of one cell is fixed to be 4.

[b] This space contains only *DARTS-V1* architecture.

ence Foundation of China (62306291), the Zhejiang Provincial Natural Science Foundation of China under Grant No. LQ24F020029; in part by National Priorities Research Program (NPRP) grants: NPRP 8-274-2-107 from Qatar National Research Fund; in part by the National Natural Science Foundation of China under Grant Nos. U20A20220, U22A6001, 61821091, 61888102, and 61825404; in part by the Key R&D Program of Zhejiang (no. 2022C01048), in part by the Strategic Priority Research Program of the Chinese Academy of Sciences under Grant XDB44000000, and in part by the Exploratory Research Project of Zhejiang Lab, China (No. 2022PF0AN01).

References

Ahn, J., Hong, S., Yoo, S., Mutlu, O., & Choi, K. (2015). A scalable processing-in-memory accelerator for parallel graph processing. In *Proceedings of the 42nd annual international symposium on computer architecture* (pp. 105–117).

Boroumand, A., Ghose, S., Kim, Y., Ausavarungnirun, R., Shiu, E., Thakur, R., et al. (2018). Google workloads for consumer devices: Mitigating data movement bottlenecks. In *Proceedings of the twenty-third international conference on architectural support for programming languages and operating systems* (pp. 316–331).

Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., et al. (2014). Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, *42*(1), 269–284.

Chen, Y., Luo, T., Liu, S., Zhang, S., He, L., Wang, J., et al. (2014). Dadiannao: A machine-learning supercomputer. In *47th annual IEEE/ACM international symposium on microarchitecture* (pp. 609–622).

Chen, X., Xie, L., Wu, J., & Tian, Q. (2019). Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 1294–1303).

Cheng, M., Xia, L., Zhu, Z., Cai, Y., Xie, Y., Wang, Y., et al. (2017). Time: A training-in-memory architecture for memristor-based deep neural networks. In *54th ACM/EDAC/IEEE design automation conference* (pp. 1–6).

Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., et al. (2016). Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. *ACM SIGARCH Computer Architecture News*, *44*(3), 27–39.

Chua, L. (1971). Memristor-the missing circuit element. *IEEE Transactions on Circuit Theory*, *18*(5), 507–519.

Dazzi, M., Sebastian, A., Parnell, T. P., Francese, P. A., Benini, L., & Eleftheriou, E. (2021). Efficient pipelined execution of CNNs based on in-memory computing and graph homomorphism verification. *IEEE Transactions on Computers*, *70*(6), 922–935.

Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, *20*(1), 1997–2017.

Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, *1*(1), 47–62.

He, X., Zhao, K., & Chu, X. (2021). AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, *212*, Article 106622.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Hu, M., Li, H., Wu, Q., & Rose, G. S. (2012). Hardware realization of BSB recall function using memristor crossbar arrays. In *Design automation conference* (pp. 498–503).

Huang, S., Ankit, A., Silveira, P., Antunes, R., Chalamalasetti, S. R., El Hajj, I., et al. (2021). Mixed precision quantization for ReRAM-based DNN inference accelerators. In *2021 26th Asia and South Pacific design automation conference* (pp. 372–377).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach, & D. M. Blei (Eds.), *Vol. 37*, *Proceedings of the 32nd international conference on machine learning* (pp. 448–456).

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., et al. (2014). Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on multimedia* (pp. 675–678).

Jiang, W., Lou, Q., Yan, Z., Yang, L., Hu, J., Hu, X. S., et al. (2020). Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Transactions on Computers*, *70*(4), 595–605.

Jiang, W., Sha, E. H., Zhuge, Q., Yang, L., Dong, H., & Chen, X. (2021). On the design of minimal-cost pipeline systems satisfying hard/soft real-time constraints. *IEEE Transactions on Emerging Topics in Computing*, *9*(1), 24–34.

Jo, S. H., Chang, T., Ebong, I., Bhadviya, B. B., Mazumder, P., & Lu, W. (2010). Nanoscale memristor device as synapse in neuromorphic systems. *Nano Letters*, *10*(4), 1297–1301.

Keckler, S. W., Dally, W. J., Khailany, B., Garland, M., & Glasco, D. (2011). GPUs and the future of parallel computing. *Vol. 31*, In *IEEE micro* (5), (pp. 7–17).

Kim, Y., Zhang, Y., & Li, P. (2015). A reconfigurable digital neuromorphic processor with memristive synaptic crossbar for cognitive computing. *ACM Journal on Emerging Technologies in Computing Systems*, *11*(4), 1–25.

Krestinskaya, O., James, A. P., & Chua, L. O. (2019). Neuromemristive circuits for edge computing: A review. *IEEE Transactions on Neural Networks and Learning Systems*, *31*(1), 4–23.

Li, B., Shan, Y., Hu, M., Wang, Y., Chen, Y., & Yang, H. (2013). Memristor-based approximated computation. In *International symposium on low power electronics and design* (pp. 242–247).

Li, B., Wang, Y., Wang, Y., Chen, Y., & Yang, H. (2014). Training itself: Mixed-signal training acceleration for memristor-based neural network. In *19th Asia and South Pacific design automation conference* (pp. 361–366).

Liu, H., Simonyan, K., & Yang, Y. (2019). DARTS: Differentiable architecture search. In *International conference on learning representations* (pp. 4561–4574).

Loni, M., Mohan, A., Asadi, M., & Lindauer, M. (2023). Learning activation functions for sparse neural networks. CoRR. arXiv:2305.10964.

Loni, M., Mousavi, H., Riazati, M., Daneshtalab, M., & Sjödin, M. (2022). TAS: Ternarized neural architecture search for resource-constrained edge devices. In C. Bolchini, I. Verbauwhede, & I. Vatajelu (Eds.), *2022 design, automation & test in Europe conference & exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022* (pp. 1115–1118).

Loni, M., Zoljodi, A., Majd, A., Ahn, B. H., Daneshtalab, M., Sjödin, M., et al. (2022). FastStereoNet: A fast neural architecture search for improving the inference of disparity estimation on resource-limited platforms. *IEEE Transactions on Systems, Man, and Cybernetics*, *52*(8), 5222–5234.

Lyu, B., Hamdi, M., Yang, Y., Cao, Y., Yan, Z., Li, K., et al. (2023). Efficient spectral graph convolutional network deployment on memristive crossbars. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *7*(2), 415–425.

Lyu, B., Wang, S., Wen, S., Shi, K., Yang, Y., Zeng, L., et al. (2023). AutoGMap: Learning to map large-scale sparse graphs on memristive crossbars. *IEEE Transactions on Neural Networks and Learning Systems*, 1–11.

Lyu, B., Wen, S., Shi, K., & Huang, T. (2021). Multiobjective reinforcement learning-based neural architecture search for efficient portrait parsing. *IEEE Transactions on Cybernetics*.

Lyu, B., Wen, S., Yang, Y., Chang, X., Sun, J., Chen, Y., et al. (2023). Designing efficient bit-level sparsity-tolerant memristive networks. *IEEE Transactions on Neural Networks and Learning Systems*, 1–10.

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., & Modha, D. S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm. In *IEEE custom integrated circuits conference* (pp. 1–4).

Mousavi, H., Loni, M., Alibeigi, M., & Daneshtalab, M. (2023). DASS: differentiable architecture search for sparse neural networks. *ACM Transactions on Embedded Computing Systems*, *22*(5s), 105:1–105:21.

Peng, X., Huang, S., Jiang, H., Lu, A., & Yu, S. (2021). DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *40*(11), 2306–2319.

Ronen, R., Eliahu, A., Leitersdorf, O., Peled, N., Korgaonkar, K., Chattopadhyay, A., et al. (2022). The bitlet model: A parameterized analytical model to compare PIM and CPU systems. *ACM Journal on Emerging Technologies in Computing Systems*, *18*(2), 1–29.

Shafiee, A., Nag, A., Muralimanohar, N., Balasubramonian, R., Strachan, J. P., Hu, M., et al. (2016). ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, *44*(3), 14–26.

Song, L., Zhuo, Y., Qian, X., Li, H., & Chen, Y. (2018). GraphR: Accelerating graph processing using ReRAM. In *IEEE international symposium on high performance computer architecture* (pp. 531–543).

Strukov, D. B., Snider, G. S., Stewart, D. R., & Williams, R. S. (2008). The missing memristor found. *Nature*, *453*(7191), 80–83.

Sun, H., Wang, C., Zhu, Z., Ning, X., Dai, G., Yang, H., et al. (2022). Gibbon: Efficient co-exploration of NN model and processing-in-memory architecture. In C. Bolchini, I. Verbauwhede, I. Vatajelu (Eds.), *2022 design, automation & test in Europe conference & exhibition, DATE 2022, Antwerp, Belgium, March 14-23, 2022* (pp. 867–872).

Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, *24*(2), 394–407.

Wen, S., Hu, R., Yang, Y., Huang, T., Zeng, Z., & Song, Y. (2019). Memristor-based echo state network with online least mean square. *IEEE Transactions on Systems, Man, and Cybernetics*, *49*(9), 1787–1796.

Wen, S., Wei, H., Yan, Z., Guo, Z., Yang, Y., Huang, T., et al. (2019). Memristor-based design of sparse compact convolutional neural network. *IEEE Transactions on Network Science and Engineering*, *7*(3), 1431–1440.

Wen, S., Wei, H., Yang, Y., Guo, Z., Zeng, Z., Huang, T., et al. (2021). Memristive LSTM network for sentiment analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, *51*(3), 1794–1804.

Wen, S., Wei, H., Zeng, Z., & Huang, T. (2018). Memristive fully convolutional network: An accurate hardware image-segmentor in deep learning. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*(5), 324–334.

Wen, S., Xie, X., Yan, Z., Huang, T., & Zeng, Z. (2018). General memristor with applications in multilayer neural networks. *Neural Networks*, *103*, 142–149.

Xia, L., Li, B., Tang, T., Gu, P., Chen, P., Yu, S., et al. (2018). MNSIM: simulation platform for memristor-based neuromorphic computing system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(5), 1009–1022.

Xie, L., Chen, X., Bi, K., Wei, L., Xu, Y., Wang, L., et al. (2021). Weight-sharing neural architecture search: A battle to shrink the optimization gap. *ACM Computing Surveys*, *54*(9), 1–37.

Yakopcic, C., Alom, M. Z., & Taha, T. M. (2016). Memristor crossbar deep network implementation based on a convolutional neural network. In *International joint conference on neural networks* (pp. 963–970).

Yakopcic, C., Alom, M. Z., & Taha, T. M. (2017). Extremely parallel memristor crossbar architecture for convolutional neural network implementation. In *International joint conference on neural networks* (pp. 1696–1703).

Yan, Z., Chen, J., Hu, R., Huang, T., Chen, Y., & Wen, S. (2020). Training memristor-based multilayer neuromorphic networks with SGD, momentum and adaptive learning rates. *Neural Networks*, *128*, 142–149.

Yang, X., Yan, B., Li, H., & Chen, Y. (2020). ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration. In *IEEE/ACM international conference on computer aided design* (pp. 92:1–92:9). IEEE.

Yang, L., Yan, Z., Li, M., Kwon, H., Lai, L., Krishna, T., et al. (2020). Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *57th ACM/IEEE design automation conference* (pp. 1–6).

Zhang, J., Yang, H., Chen, F., Wang, Y., & Li, H. (2019). Exploring bit-slice sparsity in deep neural networks for efficient ReRAM-based deployment. In *Fifth workshop on energy efficient machine learning and cognitive computing* (pp. 1–5). IEEE.

Zhou, H., Yang, M., Wang, J., & Pan, W. (2019). BayesNAS: A Bayesian approach for neural architecture search. In *36th international conference on machine learning* (pp. 7603–7613).

Zhu, Z., Sun, H., Qiu, K., Xia, L., Krishnan, G., Dai, G., et al. (2020). MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems. In *Proceedings of the 2020 on great lakes symposium on VLSI* (pp. 83–88).

Zoph, B., & Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International conference on learning representations* (pp. 1–16).

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697–8710).