Gibbon: An Efficient Co-Exploration Framework of NN Model and Processing-In-Memory Architecture

Hanbo Sun, Zhenhua Zhu, Chenyu Wang, Xuefei Ning, Guohao Dai, *Member, IEEE*, Huazhong Yang, *Fellow, IEEE*, and Yu Wang, *Fellow, IEEE*

***Abstract*—The memristor-based processing-in-memory (PIM) architectures have been proven to be a potential architecture to store enormous parameters and execute the complicated computations of deep neural networks (DNNs) efficiently. Existing PIM studies focus on designing high energy-efficient hardware architecture and algorithm-hardware co-optimization for better performance. However, the impacts of the algorithms and hardware architectures on the performance intersect with each other. Only optimizing the algorithms or the hardware architectures cannot realize the optimal design. Therefore, the co-exploration of NN models and PIM architecture is necessary. However, for one thing, the co-exploration space size of NN models and PIM architectures is extremely huge, and is challenging to search. For another, during the co-exploration process, time-consuming PIM simulators are needed to evaluate various design candidates and pose a heavy time burden. To tackle these problems, we propose an efficient co-exploration framework of NN models and PIM architectures, named *Gibbon*. In *Gibbon*, the co-exploration space is carefully designed to adapt both NN models and PIM architectures. Besides, in order to improve search efficiency, we propose an evolutionary search algorithm with adaptive parameter priority (ESAPP). In addition, *Gibbon* introduces a multilevel joint simulator to alleviate the problem of time-consuming evaluation. The experimental results show that the proposed co-exploration framework can find better NN models and PIM architectures than existing studies in only six GPU hours (9.8×–48.2× speed-up). At the same time, *Gibbon* can improve the accuracy of co-design results by 15.3% and reduce the energy–delay-product (EDP) by 5.96× compared with existing work.**

***Index Terms*—Hardware and software co-exploration, neural architecture search (NAS), neural network (NN) accelerator, processing-in-memory (PIM) architectures.**

I. Introduction

NOWADAYS, deep neural networks (DNNs) have made great breakthroughs in various fields, such as computer vision [1] and natural language processing [2], and are becoming the mainstream method to solve problems. However, DNNs introduce explosive parameters and complicated computations, causing high energy consumption and long computing time. This problem is challenging for the memory storage and the executing devices, hindering its deployment and applications.

Emerging memristor-based processing-in-memory (PIM) architectures have shown great potential to accelerate neural network (NN) computing. Based on the memristor, the PIM architecture can perform in-situ matrix–vector-multiplications (MVMs) computing and reduce redundant data movement. Therefore, PIM-based NN accelerators can improve the energy efficiency of NN computing by two to three orders of magnitude over GPU and CMOS ASIC solutions [3], [4], [5], [6], [7].

Designing the PIM hardware architecture and algorithm-hardware co-optimization for target NN models (e.g., pruning and mapping) [7], [9] are two major research directions in the PIM field. However, these studies neglect the complex interplay between the NN model hyperparameters (e.g., kernel size and network depth) and the PIM architecture design hyperparameters (e.g., Analog–Digital-Converter resolution and crossbar size) on the hardware performance and accuracy. Due to the complex interplay, the optimal NN models for different PIM architectures are different and vice versa. For instance, as shown in Fig. 1, when the NN model hyperparameters are fixed (the same column), the difference between PIM architectures can introduce up to 91.95% energy consumption variation and 2.72% accuracy variation. At the same

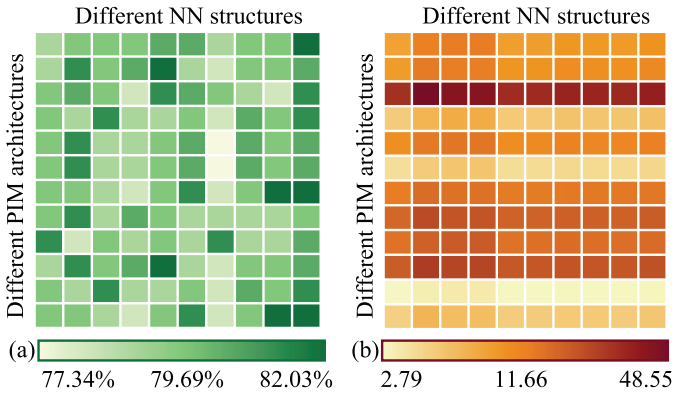


Fig. 1. (a) PIM-based NN accuracy and (b) energy consumption (mJ) of different NN structures and PIM architectures on the CIFAR-10 dataset [8].

Manuscript received 24 October 2022; revised 31 January 2023; accepted 18 March 2023. Date of publication 29 March 2023; date of current version 20 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61832007, Grant U19B2019, Grant U21B2031, and Grant 62104128; in part by the Tsinghua EE Xilinx AI Research Fund; in part by the Tsinghua–Meituan Joint Institute for Digital Life; and in part by the Beijing National Research Center for Information Science and Technology (BNRist). This article was recommended by Associate Editor X. Lin. *(Corresponding authors: Xuefei Ning; Yu Wang.)*

Hanbo Sun, Zhenhua Zhu, Chenyu Wang, Xuefei Ning, Huazhong Yang, and Yu Wang are with the Department of Electronic Engineering and the Beijing National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: foxdoraame@gmail.com; yu-wang@tsinghua.edu.cn).

Guohao Dai is with the Qing Yuan Research Institute, Shanghai Jiao Tong University, Shanghai 200030, China.

Digital Object Identifier 10.1109/TCAD.2023.3262201

time, while the PIM architecture design hyperparameters are fixed (the same row), different NN models cause a 28.31% energy consumption difference and 3.45% accuracy difference. Besides, for different optimization targets, e.g., accuracy and energy consumption, the optimal NN model and PIM architecture designs are also different. In Fig. 1(a), the optimal design for the best accuracy is in the first row and the last column, while the optimal design for the lowest energy consumption is in the second row from the bottom and the first column. Therefore, co-designing the NN models and PIM architectures is vital to ensure high accuracy and low-energy consumption simultaneously for PIM-based NN accelerators.

However, manual co-design of the NN models and PIM architectures is unrealistic due to the vast search space. For instance, the typical search space size (i.e., the total number of all possible candidates) of NN models[1] can achieve up to 3.3×10^{54}. Moreover, the search space size of PIM architectures[2] can raise to 1.3×10^{30}. Therefore, the search space size of the NN models and PIM architectures reaches 4.3×10^{84}, which is unrealistic for manual co-design.

Researchers have proposed neural architecture search (NAS) to optimize the NN structures automatically [10], [11]. Compared with manually designed NN structures, NAS can find NN structures with higher accuracy and lower computation cost [12]. Inspired by the great success of NAS, NACIM [13], UAE [14], and NAS4RRAM [15] proposed PIM-oriented NAS framework to co-explore the NN models and the PIM architectures automatically. These studies introduce the PIM architecture design hyperparameters into the NN model search space. Furthermore, they utilize PIM simulators, e.g., MNSIM [16] and NeuroSim [17], to evaluate the PIM-based NN accuracy and other hardware performance. However, these studies suffer from poor hardware performance and extremely long search time cost. To be specific, we summarize the weaknesses of existing PIM-oriented NAS methods into the following three main parts.

First, these studies introduce PIM architecture design hyperparameters into the typical NN model search space. However, the typical NN model search space lacks consideration of the PIM hardware characteristics. For example, the computing units in PIM architectures are crossbars, and crossbars are always even sized. However, the convolutional kernels in the typical NN model search space are always odd sized. Incompatible search space for the PIM architectures leads to low PIM hardware resource utilization. For instance, NACIM searches for the optimal design based on the VGG [18]-like NN model search space, which is not designed for PIM architectures. As a result, the PIM hardware resource utilization of the crossbars is only 53.1%.

Second, these studies introduce various PIM architecture design hyperparameters, e.g., the crossbar size and memristor precision, into the co-exploration space, resulting in an explosive expansion of the search space size. As mentioned in the former example, the search space size increases from 3.3×10^{54} to 4.3×10^{84} after taking the PIM architecture design hyperparameters into account. The expansion of the search space size makes it more difficult for existing search strategies to discover the optimal design candidate.

Finally, PIM-based simulators are an indispensable part of the co-exploration process of the NN models and PIM architectures. Simulators are utilized to evaluate the accuracy and hardware performance of design candidates. However, existing PIM-based simulators are highly time consuming, resulting in an unacceptable time burden for the PIM-oriented NAS. For example, in existing PIM-oriented NAS frameworks, NeuroSim and MNSIM are the two main-stream simulators for design candidate evaluation. These simulators need around 10 min to evaluate the accuracy and hardware performance for each design candidate. Time-consuming simulation poses a heavy time-cost burden to the NAS process, in which thousands of design candidates need performance evaluation. For instance, a typical PIM-oriented NAS framework needs to evaluate 3000 design candidates. Evaluating all the design candidates based on PIM-based simulators takes over 21 days.

To address these problems, we propose an efficient co-exploration framework for NN models and PIM architectures in this article, named *Gibbon*. *Gibbon* consists of three main components: 1) a PIM-oriented search space to achieve better accuracy and hardware performance; 2) an evolutionary search algorithm to improve the search efficiency; and 3) a multilevel joint simulator to alleviate the problem of heavy time-cost burden. Owing to the well-designed search space, efficient search strategy, and efficient and precise simulator, *Gibbon* can reduce the search time from hundreds of GPU hours to only several GPU hours. Moreover, *Gibbon* can find better NN models and PIM architectures with higher accuracy and better hardware performance than existing PIM-oriented NAS methods. The main contributions of this article can be summarized as follows.

1) We propose a PIM-oriented search space for NN models and PIM architectures. We first introduce group convolution and convolution with even sized kernels into the PIM-oriented NN model search space. Besides, we propose a PIM friendly NN topology.
2) We propose an evolutionary search algorithm with adaptive parameter priority (ESAPP) to realize high search efficiency. ESAPP assigns different priorities to different search hyperparameters to construct small subsearch spaces. Consequently, ESAPP can reduce the search time by ∼90%.
3) We propose a recurrent NN (RNN) [19]-based accuracy and hardware performance predictor. The proposed predictor can realize **72**× speed up with only 2.6% accuracy prediction error compared with typical PIM simulators. Based on the efficient and precise predictor, we construct a multilevel joint simulator and utilize the predictor to filter 95% of design candidates.
4) Experimental results show that compared with existing state-of-the-art PIM-oriented NAS framework, *Gibbon* can achieve **9.8**×–**48.2**× speed-up, and only takes *six* GPU hours to find the optimal design. As for the

[1]The search space is constructed by NN models with up to 30 convolutional layers. Each convolution layer can be configured with different kernel sizes, group numbers, output channel numbers, etc.

[2]We modify the crossbar size, DAC/ADC resolution, memristor precision, etc., of a base PIM architecture to generate new PIM architecture candidates.

accuracy and hardware performance, *Gibbon* can realize up to 15.3% NN accuracy improvement and 5.96× energy–delay-product (EDP) reduction.

5) Based on the results of *Gibbon*, we provide several insights on the correlations of the NN model and PIM architecture, which are helpful to guide the co-design of the NN models and PIM architectures in the future.

The remainder of this article is organized as follows. Section II introduces the DNN and PIM architecture basics to facilitate the understanding of the proposed co-exploration framework. Besides, Section II also discusses related work of existing NAS and hardware-oriented NAS methods. Section III gives an overview of the co-exploration framework. Sections IV–VI explain the detailed design technique of the proposed search space, search strategy, and multilevel joint simulator, respectively. The experimental results and comparison with other existing work are shown in Section VII, and Section VIII lists some insights for the co-design of the NN models and PIM architecture in the future. Finally, we conclude this article in Section IX.

II. Preliminary and Related Work

A. Deep Neural Network

DNNs can be represented by a directed acyclic graph (DAG). Each vertex in the DAG denotes an operation, e.g., convolution, fully connected operation, and batch normalization. Moreover, the topology of the DAG represents the data dependency among different operations. Convolutional operations are commonly used operations in DNNs. A typical convolutional operation can be described as follows:

$$A_o(x, y, c) = \sum_{i=0}^{K-1} \sum_{j=0}^{K-1} \sum_{k=0}^{C_{in}-1} A_{\text{in}}(x+i, y+j, k) w_c(i, j, k) \quad (1)$$

where A_o and A_{in} denote the 3-D output and input feature map, respectively. And w_c is a 3-D weight matrix corresponding to the cth output channel with the size of $K \times K \times C_{\text{in}}$. The key factors affecting accuracy and hardware performance are the topology and the hyperparameters of operations, e.g., kernel sizes and the number of output channels.

B. Processing-In-Memory Architecture

Emerging memristors (e.g., Resistive Random Access Memory [20], Phase Change Memory [21], and Magnetic Random Access Memory [22]) provide an alternative solution to realize high energy efficiency NN accelerators. Multiple memristors can construct the crossbar structure. When applying the input voltages $\mathbf{V}$ on the word lines of the crossbar and mapping the weights $\{g_{m,n}\}$ as the cell conductance of the crossbar, we can acquire the output current $\mathbf{I}$ on the bit lines. According to Kirchhoff's laws, the relationship of $\mathbf{I}$ and $\mathbf{V}$ can be described as follows:

$$I_m = \sum_{k=0}^{N-1} V_k g_{m,k} \quad (2)$$

where N is the number of crossbar word lines, and V_k and I_m represent the voltage on the kth word line and the current on the mth bit line, respectively. Therefore, the MVMs can be performed in crossbars. In consideration of MVMs being executed in the analog domain, digital-to-analog converters (DACs) and analog-to-digital converters (ADCs) are important components in crossbars. Existing studies have proposed various memristor-based PIM architecture and realize 2–3 orders of magnitude energy efficiency improvement compared with GPU and CMOS-based ASIC solutions [3], [4], [5], [6], [23].

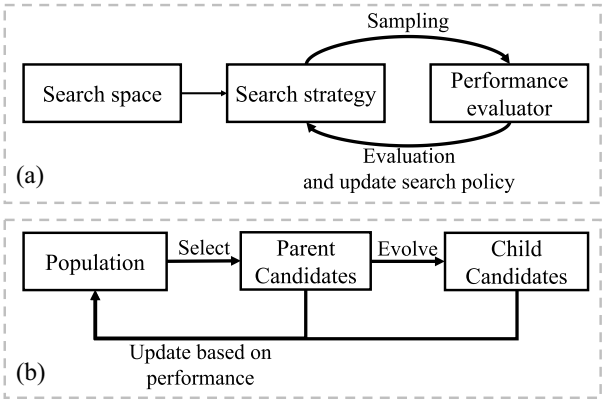


Fig. 2. (a) Typical NAS process and (b) process of evolutionary algorithms.

C. Neural Architecture Search

NAS is proposed to automatically design well-performing NNs [10], [24], [25]. In general, NAS consists of three major components: 1) search space; 2) search strategy; and 3) performance evaluator, and the typical NAS process is shown in Fig. 2(a). The search space contains all the possible design candidates for the NN structure. The search strategy samples design candidates and updates its search policy based on the evaluation results to sample candidates with better performance. Moreover, after several iterations, the search strategy will find the best search policy and produce the optimal NN structure. As for the performance evaluator, it receives design candidates and outputs the corresponding performance.

The search strategy is an essential component in the NAS process. A typical kind of search strategies is the evolutionary algorithm [26], [27], [28], [29]. Evolutionary algorithms usually maintain one population group, a set of design candidates, and find the optimal design candidate through population evolution. Population evolution conducts three major steps, as shown in Fig. 2(b).

1) *Parent Selection:* Select parent design candidates from the original population group.
2) *Child Generation:* Derive child design candidates from the selected parent candidates by the genotype "mutation" and "crossover."
3) *Population Update:* Update the population based on the performance of the selected parents and derived child candidates.

D. Hardware-Oriented NAS-Related Work

Recently, researchers have applied NAS methods in the NN structure and accelerator architecture co-design [30]. For

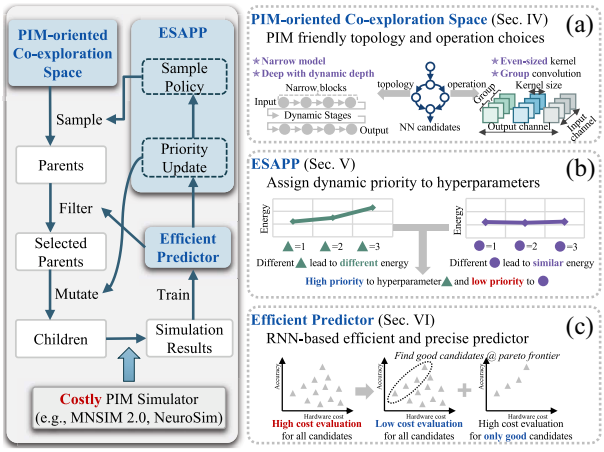


Fig. 3. Co-exploration framework overview (left part). *Gibbon* contains three key components: (a) PIM-oriented co-exploration space, (b) ESAPP, and (c) RNN-based accuracy and hardware performance predictor.

instance, NAAS [31] and NASAIC [32] adopt NAS to find optimal NN structure and accelerator architecture design for FPGA- and ASIC-based accelerators, respectively. They achieve better accuracy and hardware performance with lower hardware resource cost. There are also several PIM-oriented NAS methods in the PIM field [13], [14], [15]. NAS4RRAM [15] proposes a PIM-aware NAS framework to find an NN with the highest accuracy with the area constraints. However, NAS4RRAM searches only for NN structures without considering the design hyperparameters of PIM architectures. NACIM [13] and UAE [14] introduce hardware design hyperparameters into the search space. Therefore, NACIM and UAE can co-explore NN structures and PIM architectures. However, NACIM utilizes a time-consuming PIM simulator (i.e., NeuroSim [17]) as the performance evaluator, resulting in a tremendous search time cost ($\sim$ 59-GPU hours). UAE adopts a more complicated evaluation strategy for the performance evaluator to better model the PIM architecture, causing an even more severe time cost ($\sim$ 154-GPU hours).

III. Co-Exploration Framework Overview

The proposed co-exploration framework for NN models and PIM architectures (*Gibbon*) is shown in Fig. 3. As shown in this figure, *Gibbon* consists of three main components: 1) the PIM-oriented search space for NN models and PIM architectures; 2) the ESAPP; and 3) the multilevel joint simulator with an RNN-based performance predictor.

The PIM-oriented search space contains design candidates for NN models and PIM architectures. Each candidate specifies the design choices related to both the NN models and the PIM architectures. Based on the PIM-oriented search space, each iteration of the search process goes as shown in the left of Fig. 3. First, ESAPP samples multiple parent design candidates (**parents**) and sends them to the RNN-based performance predictor. Based on the prediction results, we use accuracy and hardware performance as the evaluation metric to filter out $\sim$95% of the total sampled parent design candidates and output the better 5% candidates as selected parent design candidates (**selected parents**).

Afterward, ESAPP mutates the selected parent design candidates to get new design candidates (**children**), where the mutations are conducted according to the priorities of search hyperparameters. Then, the child design candidates are evaluated by the accurate but time-consuming PIM simulator (in this article, we use MNSIM [33]). Finally, to maintain a precise prediction of the proposed performance predictor, we utilize the evaluation results from the accurate but time-consuming PIM simulator **to fine tune** and update the predictor. At the same time, ESAPP analyzes the evaluation results to update the priorities of search hyperparameters. *Gibbon* repeats these steps until the search converges and outputs the optimal design candidate of the NN models and PIM architectures.

IV. PIM-Oriented Search Space for NN and PIM Co-Exploration

The proposed PIM-oriented search space for the NN model and PIM architecture co-exploration is the Cartesian product of the following two search spaces: 1) the NN structure search space and 2) the PIM-related hardware search space.

The PIM-related hardware search space is designed to explore the hardware configurations of PIM architectures. We adopt the PIM architecture proposed in MultiPrecision [6] as our infrastructure, for it can achieve higher equivalent energy efficiency with nearly no accuracy loss. Our infrastructure has various hardware configurations, e.g., crossbar sizes, ADC/DAC resolutions, memristor precision, and the number of activated word lines and bit lines at one time, and so on. The choices of these hardware configurations construct the PIM architecture search space, which is one part of the PIM-related hardware search space.

The other part of the PIM-related hardware search space is the quantization search space. The quantization search space contains the weight quantization configurations and the activation quantization configurations of each operation as design hyperparameters. The quantization configurations are related to the memristor precision and ADC/DAC resolutions in the PIM architectures. In our quantization search space, there can be different quantization configurations for weights and activations of different operations. In this way, we can achieve significant performance improvements to PIM-based NN accelerating systems [6].

As for the NN structure search space, we carefully design the NN structure search space to obtain a compatible and PIM-friendly co-exploration space. As mentioned in Section II-A, NN models can be described as a DAG, where each vertex denotes an operation, and each edge represents the data dependency among the operations. The topology (the edges) and the operation choices (the vertex) are various for different NN model candidates. Therefore, we split the NN structure search space into two parts: 1) the DAG topology search space and 2) the operation choice search space.

A. DAG Topology Search Space

We refer to NAS4RRAM [15] to construct the baseline DAG topology. As shown in the left of Fig. 4(a), in the baseline topology search space, there is a head module, a tail module,

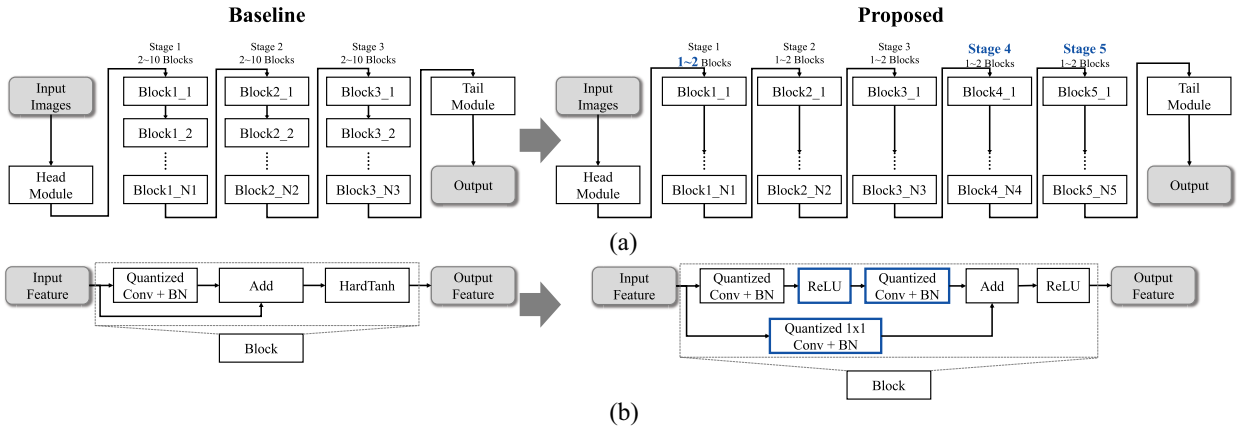


Fig. 4. DAG topology comparison between the baseline search space (left, referring to NAS4RRAM [15]) and the proposed search space (right). (a) Overall structure. (b) Structure of the basic block.

and blocks in three stages in each design candidate. And in each stage, there are two to ten basic blocks. The head module consists of a convolutional operation and a Rectified Linear Unit (ReLU) to generate low-level features for input images. Besides, the tail module contains a sequence of fully connected operations to transfer features into the final prediction outputs. As for each stage, the first and only first block downsamples the input features, and all blocks in the same stage perform feature extraction in the same feature map size. In Fig. 4(a), N_i denotes the number of basic blocks in the ith stage, and is also one design hyperparameter of the baseline topology search space. The left of Fig. 4(b) shows the structure of the basic block in the baseline topology search space. Inspired by ResNet [34], there is a skip connection in the basic block. The two inputs of the **Add** operation are the input and output of the convolutional operation, which may have different channel numbers. However, the **Add** operation requires the same channel numbers of the two inputs. To address this problem, NAS4RRAM extends the channel number of the smaller one to that of the bigger one by filling in zeros before the add operation.

However, the baseline topology search space is not designed for PIM architectures and is incompatible with the PIM hardware, causing hardware performance reduction. On the one hand, the baseline topology search space has only three stages and up to ten blocks in each stage. Three stages lead to poor feature representation. Moreover, too many blocks in the same stage, especially, in the early stage, result in high computation amounts and poor hardware performance. Taking VGG [18] and ResNet [34] as examples, there are at least five stages beside the head and tail modules to ensure rich feature representation. And in the early stage, the size of the feature maps is bigger than that in the later stages. Therefore, there will be higher computation amounts for the same basic blocks in the early stage. Too many blocks in the early stage will aggravate this phenomenon and harm the hardware performance. On the other hand, the channel extending operation before the add operation causes channel deviation and low classification accuracy. Nevertheless, there is a linear relationship between the two input features of the add operation. In light of the add operation being linear, we can transfer the convolutional operation, add operation, and skip connection as an equivalent convolutional operation. That means the skip connection in the basic blocks is entirely ineffective.

To address these problems, we propose a new topology search space as shown in the right of Fig. 4. The proposed topology search space contains the same head and tail modules in the baseline topology search space. Unlike the baseline topology search space, the proposed topology search space has five stages and only one to two basic blocks in each stage. Owing to more stages, design candidates in the proposed topology search space can acquire rich feature representation and achieve better classification accuracy. At the same time, the proposed topology search space can reduce the computation amounts in the early stages by reducing the number of basic blocks.

As shown in the right of Fig. 4(b), we construct new basic blocks in the proposed topology search space. To address the channel deviation problem, we add an extra pointwise convolutional operation (i.e., a convolutional operation with 1×1 kernel) in the original skip connection branch [35]. The extra pointwise convolutional operation in the skip connection branch performs channel alignment and can avoid the channel deviation problem. To address the problem of the entirely ineffective skip connection, we add a ReLU in the main branch to introduce nonlinearity. Besides the ReLU operation, we introduce an extra convolutional operation in the main branch for better feature representation. In PIM architectures, the input values are represented by the voltage in the word lines and are always positive. Therefore, we replace the **HardTanH** operations (the input values may be negative) with **ReLU** operations (the input values are always positive) in the proposed topology search space.

To verify the advantages of the proposed topology search space, we train and evaluate the representative design candidates in the baseline topology search space and the proposed search space. For the representative design candidate in the baseline topology search space, the number of basic blocks in each stage is 5. For the candidate in the proposed topology search space, the number of basic blocks in each stage is one.

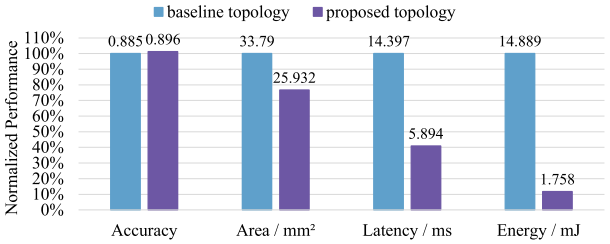


Fig. 5. PIM-based NN accuracy and hardware performance comparison between the representative design candidates.

Moreover, every convolutional operation in both representative design candidates has the same input and output channel numbers and kernel sizes. And there are both 15 convolutional operations in the above two representative design candidates. The accuracy and hardware performance comparison of the two design candidates is shown in Fig. 5. We can see that the representative design candidate in the proposed topology search space achieves better classification accuracy and can reduce the area, latency, and energy consumption by around 25%, 60%, and 90%, respectively. In a word, the proposed topology search space can reach better classification accuracy under lower hardware resource cost.

B. Operation Choice Search Space

We construct the operation choice search space to contain all possible operation choices for every vertex in the DAG. Existing NN model and PIM architecture co-exploration methods adopt the operation choice search space in the typical NAS methods [10], [25], which is not designed for PIM architectures. The operation choices in these search spaces are not compatible and friendly to PIM architectures, causing poor hardware performance. On the one hand, the numbers of rows and columns of the crossbars (the basic computing units in PIM) are always the power of two, e.g., 64, 128, and 256 [36]. In this way, the word line and the bit line addresses can be encoded and decoded based on binary encoding. However, the convolutional operations in typical NAS search spaces are always with odd-sized kernels [25], such as 1×1 and 3×3 kernels. Mapping the weights of odd-sized kernels onto the even-sized memristor crossbars deteriorates the hardware resource utilization. On the other hand, limited by the current upper bound of the bit lines, only part of the word lines in crossbars can be activated simultaneously. We set L_{wl} to represent the maximum number of the activated word lines at the same time. To support long input vectors, we split them into subvectors, where the length of each subvector is not bigger than L_{wl}. Afterward, we gather the computing results of every input subvector and get the final output vectors. Splitting the input vectors harms the hardware performance of the PIM architectures. For example, if we split the input vector into two subvectors, we need to perform analog-to-digital conversions twice. On the contrary, if we do not split the input vector, we only need to do the conversions once. Furthermore, the energy consumption of analog-to-digital conversions accounts for around 70% of the overall energy consumption [6]. Therefore, reducing the number of subvectors is critical to achieving better hardware performance. However, typical NAS search spaces neglect this characteristic of PIM architectures, leading to poor hardware performance.

To tackle these problems, we propose a new operation choice search space, which is compatible and friendly to PIM architectures. First, we explore the feasibility of even-sized convolutional kernels. We can improve hardware resource utilization by introducing convolutional operations with even-sized kernels. However, convolutional operations with even-sized kernels lead to feature shifting problem [37]. For example, in an NN where all convolutional operations are with 2×2 kernels, the receptive field only contains the lower right area and lacks features from other areas. Wu et al. [37] proposed a symmetric padding method to alleviate the feature shifting problem. The symmetric padding method splits input feature maps into four subsets, and assigns the four subfeature-maps with up, down, left, and right padding, respectively. However, the symmetric padding method introduces four different padding types in the same input feature maps. Different padding types make data at the same location on the input feature maps be computed in different clock periods, which is hard to implement on PIM architectures.

To address this problem, we propose the statistical equilibrium padding (SEP) method. Unlike the symmetric padding method, the proposed SEP method assigns different padding types in different input feature maps. Furthermore, for each input feature map, there is only one type of padding pattern. SEP method assigns random extra vertical and horizontal padding to input feature maps of all convolutional operations with even-sized kernels. The extra padding in the proposed SEP method can be described as follows:

$$\begin{aligned} \text{Padding}_{\text{vertical}} &= X_v, X_v \sim B(1, 0.5) \\ \text{Padding}_{\text{horizontal}} &= X_h, X_h \sim B(1, 0.5) \end{aligned} \tag{3}$$

where $\text{Padding}_{\text{vertical}}$ and $\text{Padding}_{\text{horizontal}}$ denote the vertical and horizontal extra padding in the input feature maps, respectively. For example, $\text{Padding}_{\text{horizontal}} = 0$ represents add extra padding to the last column of the input feature maps, and $\text{Padding}_{\text{horizontal}} = 1$ represents add extra padding to the first column. Moreover, $B(1, 0.5)$ is a Bernoulli distribution with 1 as the number of independent experiments and 0.5 as the probability of success. In this way, each design candidate in the NN structure search space has statistically the same left and right, lower and upper padding to the feature maps, alleviating the feature shifting problem.

As shown in Fig. 6, we compare the PIM-based NN accuracy and hardware performance among different convolutional kernel patterns. Compared with the 3×3 kernel pattern, the 2×2 kernel pattern achieves comparable accuracy and reduces the area, latency, and energy consumption by around 40%, 60%, and 70%, respectively. Although the 1×1 kernel pattern achieves the lowest area, latency, and energy consumption, the accuracy loss is around 70% and is unacceptable for deployment. Compared with the original 2×2 kernel pattern without the SEP method, the 2×2 kernel pattern with the SEP method can improve the accuracy by around 5% without

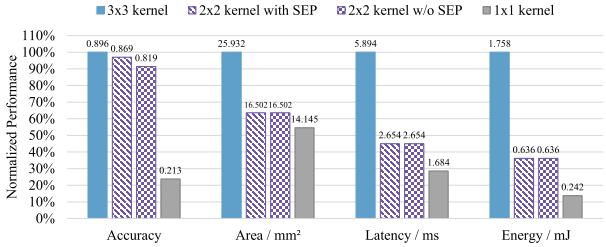


Fig. 6. PIM-based NN accuracy and hardware performance comparison among different convolutional kernel pattern, i.e., 3 × 3 kernel, 2 × 2 kernel with and without the SEP method, and 1 × 1 kernel. We select the representative design candidate in the proposed topology search space as the backbone.

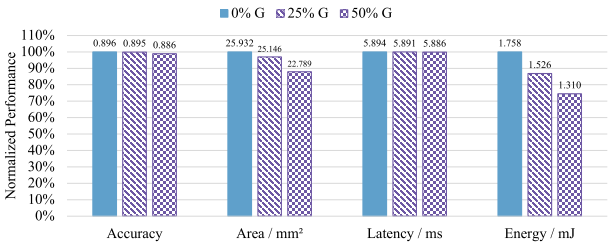


Fig. 7. PIM-based NN accuracy and hardware performance comparison among different proportions of group convolutional operations in all convolutional operations (*G* for all group convolutional operations is 2).

any hardware performance overhead. Note that we support not only the 2 × 2 kernels but also 3 × 3 and 1 × 1 in our operation choice search space. We provide a possibility to utilize 2 × 2 kernels to improve the hardware performance, e.g., area and latency.

Second, we introduce group convolutional operations into the operation choice search space. The group convolutional operation is a special kind of convolutional operation. It splits the input and output feature maps into several groups in the channel dimension. Afterward, the group convolutional operation performs typical convolutional computation for each group independently. By splitting feature maps into G groups, group convolutional operations can reduce the amount of computation and weights to $(1/G)$. Owing to the lower amounts of computations and weights, group convolutional operations are widely used in lightweight networks, e.g., MobileNet [38] and ShuffleNet [39]. Furthermore, by dividing the input feature maps into G groups in the channel dimension, grouped convolutional operations cut down the length of the input vector. Therefore, the energy consumption of analog-to-digital conversions is reduced, improving the hardware performance.

As shown in Fig. 7, we compare the PIM-based NN accuracy and hardware performance among different proportions of group convolutional operations. Baseline design candidates do not contain any group convolutional operations and are denoted by **0% G**. **25% G** represents that 25% of the total convolutional operations are group convolutional operations. And so as **50% G**. Compared to the baseline candidates, **25% G** and **50% G** can achieve comparable accuracy and latency. As for other hardware performance metrics, **50% G** decreases the area and energy consumption by around 10% and 30%, respectively. It should be noticed that not all

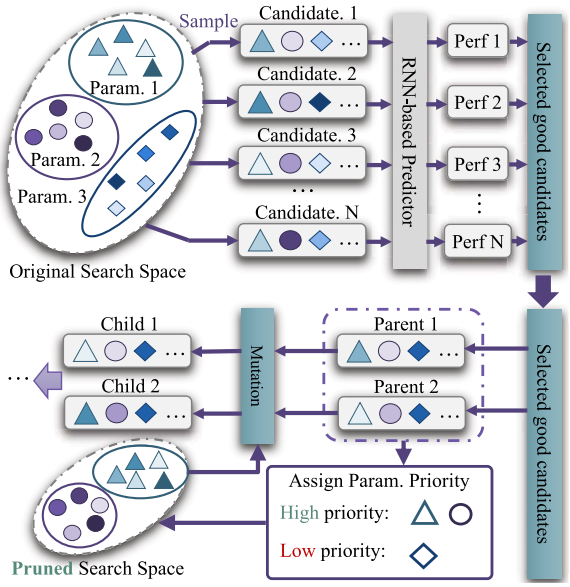


Fig. 8. Overview of the proposed ESAPP.

convolutional operations must be group convolutional operations in the proposed operation choice search space. Part of the typical convolutional operations may stay the same during the co-exploration process to obtain high PIM-based NN accuracy.

To sum up, in the operation choice search space, we explore the feasibility of even-sized convolutional kernels and group convolutional operations. The proposed operation choice search space is designed for PIM architectures and is friendly for the NN model and PIM architecture co-exploration. To the best of our knowledge, we are the first to introduce even-sized convolutional kernels and group convolutional operations in the PIM-oriented NAS field. Moreover, experimental results show that the proposed operation choices can significantly improve hardware performance with negligible accuracy loss.

V. Evolutionary Search With Adaptive Parameter Priority

The vast search space of the PIM-oriented co-exploration poses search efficiency challenges in the application of the evolutionary search. For instance, the total number of design candidates reaches up to 4.3×10^{84} in our co-exploration space of NN models and PIM architectures, resulting in hundreds of GPU hours to find the optimal design. To address this problem, we propose an ESAPP. ESAPP can be regarded as a dynamic search space pruning method to improve search efficiency. As shown in Fig. 8, it assigns different priorities to the design hyperparameters, "omitting" mutation on unimportant hyperparameters. In our experimental results, ESAPP can reduce the equivalent search space size[3] from 4.3×10^{84} to roughly 1.3×10^{22} during the search process.

ESAPP leverages the idea of search space pruning to improve search efficiency. In the children generation step

[3]The equivalent search space size is estimated by "omitting" design hyperparameters with low priorities from the original search space.

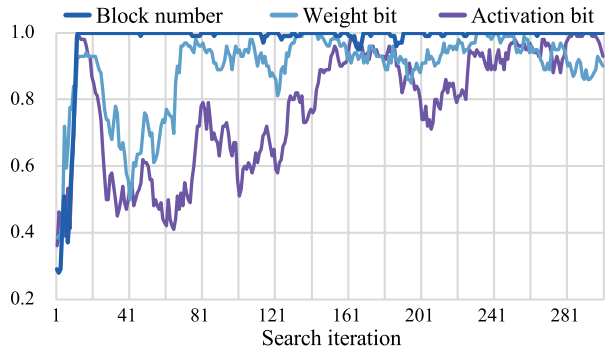


Fig. 9. Convergence of different design hyperparameters. The y-axis indicates the proportion of the optimal candidates among all candidates (the closer to 1.0, the more convergent the hyperparameter is).

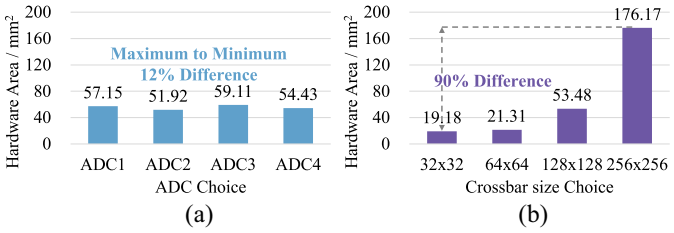


Fig. 10. Hardware area comparison under different choices of design hyperparameters. (a) and (b) Design hyperparameters for the crossbar sizes and ADC choices, respectively.

of each search iteration, ESAPP assigns a priority to each design hyperparameter and determines which hyperparameters to be mutated in this iteration. Specifically, we avoid mutating design hyperparameters with low priorities, realizing equivalent search space pruning. We assign low priorities to the following two types of design hyperparameters.

The first type contains the design hyperparameters that have been converged in the search process. As shown in Fig. 9, different design hyperparameters show different convergence curves. The block number hyperparameter converges the fastest in around the 10th iteration. While the activation bit hyperparameter converges the slowest in around the 160th iteration. Searching the easy-to-converge hyperparameters in the early search stage can help to save the search efforts and pay more attention to the difficult-to-converge hyperparameters in the later search stage. In the proposed ESAPP, converged hyperparameters are assigned low priorities to reduce the redundant search. We use *Entropy* to describe the convergences of design hyperparameters, which can be calculated as in

$$Entropy = \sum_{v_\omega \in \Omega_\omega} -f_{v_\omega} \log_2\left(f_{v_\omega}\right) \tag{4}$$

where Ω_ω is the set of all possible values for one design hyperparameter ω. f_{v_ω} represents the occurrence frequency for which the value of ω is v_ω in the selected design candidate set.

The second type contains the design hyperparameters that have little to do with the hardware performance. As shown in Fig. 10, the impact of different design hyperparameters on the hardware area performance is different. All possible ADC choices result in similar hardware areas, and the difference between the maximum and minimum is only 12%. As for the crossbar size design hyperparameter, the difference between the maximum and minimum rises to 90%. Therefore, we should pay more attention to the crossbar size design hyperparameter for better hardware area. We utilize *Intensity* to indicate the impact of design hyperparameters on the hardware performance, which can be calculated as

$$\text{Intensity} = \text{std}(\text{Perf}_\omega)/\max(\text{Perf}_\omega) \tag{5}$$

where Perf_ω represents a hardware performance vector, and we use Perf_{v_ω} to denote the element in the vector. Perf_{v_ω} means the average performance of design candidates where the value of ω is v_ω. std$(\cdot)$ and max$(\cdot)$ mean the functions to get the standard deviation and the maximum, respectively. Higher *Intensity* means that the design hyperparameter has a greater impact on the target hardware performance. Searching the design hyperparameters with low *Intensity* causes ineffective search. Therefore, ESAPP assigns low priorities to design hyperparameters with low *Intensity*.

TABLE I
EQUIVALENT SEARCH SPACE SIZE OF ASSIGNING DIFFERENT METRICS AS PRIORITIES TO THE DESIGN HYPERPARAMETERS

Priority metric	Equivalent search space size
Origin search space	4.3×10^{84}
Entropy	9.3×10^{30}
Intensity	3.9×10^{42}
Entropy & *Intensity*	1.3×10^{22}

As shown in Table I, we compare the equivalent search space size after adopting *Entropy* and *Intensity*. By assigning *Entropy* and *Intensity* as the search priorities, the equivalent search space size is reduced from 4.3×10^{84} to 9.3×10^{30} and 3.9×10^{42}, respectively. To combine *Entropy* and *Intensity*, we assign design hyperparameter priorities based on their product. In this way, only design hyperparameters with both high *Entropy* and high *Intensity* are assigned with high priorities. Based on this kind of priority metric, the equivalent search space size is reduced to 1.3×10^{22}. Compared with traditional search space pruning methods [40], ESAPP can prevent the search from being stuck into the local optimum. In traditional search space pruning methods, a hyperparameter cannot become searchable again after being pruned out, leading to the local optimum. In contrast, ESAPP dynamically adjusts the design hyperparameter priorities throughout the search.

Algorithm 1 shows the details of the proposed ESAPP. In each search iteration, ESAPP first selects design candidates with good performance, e.g., top 100 design candidates, (line 6). Then, we calculate the *Entropy* and *Intensity* based on the selected design candidate set. Afterward, we assign low priorities to design hyperparameters with low entropy and low intensity (lines 8–13). Next, we determine the mutation probability according to the design hyperparameter priorities (line 15). Finally, the mutation is executed to generate the new candidates for the next search iteration (lines 16–21).

Algorithm 1 Pseudocode of ESAPP

Input:
Population: $\{Candidate\}_{i-1}$
History priority table: *HPT*
Evaluation results of candidates: *Perf*
Output: New population: $\{Candidate\}_i$
Initialize $\{Candidate\}_i$: $\{Candidate\}_i$.init()
Select good candidates as parents:
  $Parents$ = sort_select($Perf$, $\{Candidate\}_{i-1}$)
For hyperparam. x, cal. entropy and intensity in parents:
  $Entropy_x$ = cal_e($Parents$)
  $Intensity_x$ = cal_i($Parents$)
Update *HPT*:
  $HPT = HPT$.append($\{Entropy_x, Intensity_x\}$)
Assign priorities to each hyperparam.:
  $Priority_x$ = calc_p(HPT)
Determine the mutation probability of each hyperparam.:
  $Prob_x = Priority_x / \sum_j (Priority_j)$
//*Mutate according to* $\{Prob_x\}$:
for each *parent* in *Parents* **do**
  $child$ = mutate($parent$, $\{Prob_x\}$)
  $\{Candidate\}_i$.append($parent$)
  $\{Candidate\}_i$.append($child$)
end for

VI. Multilevel Joint Simulator

As mentioned in Section III, we utilize a simulator to evaluate the hardware performance of the sampled design candidates. On the one hand, based on the evaluation results, we update the sampling strategy in the search algorithm to find the optimal design. Imprecise evaluation results lead to inaccurate sampling strategy and, thus, harm the search performance. On the other hand, thousands of design candidates will be evaluated throughout the search. As a result, design candidate evaluation takes more than 95% of the total search time consumption. Therefore, it is necessary to construct a precise and efficient simulator to evaluate the design candidates.

However, existing PIM simulators are not efficient and are unsuitable for the NN model and PIM architecture co-exploration. For example, NACIM [13] and UAE [14] adopt NeuroSim [17] as the hardware performance simulator. MNSIM 2.0 [33] is another popular and commonly used PIM simulator. NeuroSim and MNSIM 2.0 require ~10 min to evaluate the hardware performance of a design candidate [33]. Since thousands of design candidates need to be evaluated in the co-exploration process, minute-level PIM simulators will consume unacceptable search time.

To tackle this problem, we propose a multilevel joint simulator. Inside the joint simulator, we construct an efficient RNN-based predictor to predict the hardware performance of the design candidates. We utilize the efficient predictor to filter the design candidates with low hardware performance coarsely. Afterward, the PIM simulator evaluates the filtered design candidates precisely. Therefore, the number of design candidates evaluated by the time-consuming PIM simulator is greatly reduced, significantly reducing the search time. To ensure a precise prediction, we train the predictor with the precise and sufficient evaluation results generated by the PIM simulator.

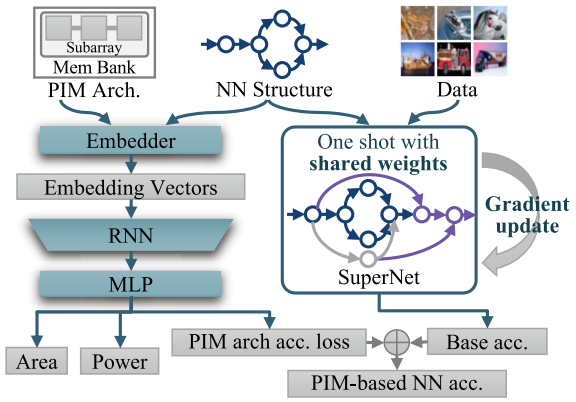


Fig. 11. Overview of our RNN-based predictor. The predictor has three key components: the design candidate embedder, the RNN-based feature extractor, and the MLP-based regressor.

A. Evaluation Metric

As shown in Fig. 11, the inputs of the efficient predictor are the description of the NN models and PIM architectures in the design candidates, such as crossbar sizes, ADC/DAC resolutions, and kernel sizes. Moreover, the outputs are the PIM-based NN accuracy and other hardware performance, such as the area, power, energy consumption, and latency. For performance predictors, ranking quality has been proven to be the critical criterion in existing NAS studies [41]. In this article, we adopt Kendall's Tau (KD) ranking correlation coefficient as the evaluation metric for the predictor. Its calculation goes as follows:

$$\mathrm{KD} = \sum_{i<j} \mathrm{sign}(y_i - y_j)\mathrm{sign}(s_i - s_j) / \binom{M}{2} \tag{6}$$

where M denotes the total number of design candidates, and y_i and s_i represent the ground-truth hardware performance and predicted hardware performance, respectively. sign(·) is the sign function. A higher KD coefficient indicates that the ranking of predictions is more similar to that of ground truth.

Compared with the typical linear correlation coefficient, the ranking correlation coefficient can better reflect the performance ranking among design candidates. Therefore, it helps us to filter the design candidates with poor hardware performance precisely. Moreover, a high linear correlation coefficient does not always mean a high ranking correlation coefficient. For instance, the linear correlation coefficient of $[0, 0.49, 0.5, 0.51, 1]$ and $[0, 0.51, 0.5, 0.49, 1]$ is 0.9992. Meanwhile, for the ranking correlation coefficient, the KD coefficient is only 0.3999, which is much lower than the linear correlation coefficient. Therefore, we should not adopt the linear correlation coefficient to measure the predictor performance.

The efficient predictor is constructed based on the RNN structure. Furthermore, as depicted in Fig. 11, the RNN-based predictor consists of three key components: 1) the design candidate embedder; 2) the feature extractor; and 3) the regressor.

B. Design Candidate Embedder

The *design candidate embedder* transforms the discrete description of the NN models and PIM architectures in the design candidates into continuous embedding vectors. The description of the design candidates is discrete. For instance, the kernel size of a convolutional operation belongs to a finite set {1, 2, 3} in our NN structure search space. And assigning 1.2 as the kernel size is invalid and meaningless. The description of the design candidates has an inherent correlation between different types of description. For example, a convolutional operation with 16 input channels and 2×2 kernels consumes 64 ($= 16 \times 2^2$) crossbar word lines. While another convolutional operation with 64 input channels and 1×1 kernels consumes 64 ($= 64 \times 1^2$) crossbar word lines too.

Taking the discrete description as the inputs makes it hard to train the predictor and is unable to reflect the description's inherent correlation. To address this problem, we construct a description embedding method to transfer the discrete description into embedding vectors. The embedding vectors are in a continuous space, helping us to train the predictor. Furthermore, different description types may have close embedding vectors, reflecting the description's inherent correlation. The description embedding process goes as follows:

$$V = CodeBook_n[D] \tag{7}$$

where D denotes the description index and V represents the output embedding vector. CodeBook$_n$ means a list of embedding vectors, each element being a *trainable* n-dimensional vector. For example, supposing the kernel size belongs to a finite set $\{1, 2, 3\}$, the description index of kernel_size $= 1$ is 0 (the index in the set). Then, the output embedding vector is the first element in the CodeBook$_n$. Moreover, for different description types, we set different *CodeBook*s. Based on the proposed description embedding method, we utilize efficient gradient-based optimization methods, e.g., stochastic gradient descent (SGD) [42], to train the predictor efficiently. In the optimization process, we also adopt gradient-based optimization on the embedding vectors of the *CodeBook*s. Therefore, we can construct the description's inherent correlation.

C. Feature Extractor

The *feature extractor* takes the former embedding vectors as the inputs and extracts the features of the design candidates. Each design candidate in our co-exploration space contains five stages, and each stage is stacked with multiple blocks. These blocks have the following characteristics. For one thing, all blocks in design candidates are isomorphic and share the same basic block structure. The description of blocks constitutes sequential data, and each element means the block input features. For another, the numbers of the blocks are various in different design candidates. RNNs [19] are commonly used network structures to process sequential data, which are able to handle variable-length data. Therefore, we propose an RNN-based feature extractor.

In the proposed RNN-based feature extractor, besides the embedding vector of the block structure and hyperparameters, we also introduce the embedding vectors of the PIM architectures as the block input features. Block input features without the embedding vectors of the PIM architectures make it difficult to model the block hardware performance, e.g., the area and energy consumption. Therefore, introducing the hardware description helps us better model the design candidates' hardware performance, especially, for the area and energy consumption. As shown in Fig. 12, we compare the KD coefficient of the feature extractors with and without the hardware description as the block input features for the area and energy consumption. Experimental results show that introducing the hardware description into the block input features improves the KD coefficient by around 10% and 5% for the area and energy consumption, respectively.

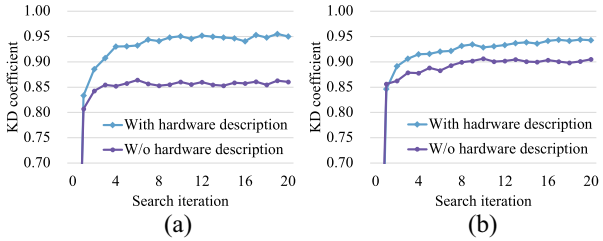


Fig. 12. KD coefficient comparison of the feature extractors with and without the hardware description for the prediction of (a) area and (b) energy consumption.

D. Regressor

The *regressor* in our predictor is a three-layer multilayer perception (MLP). It outputs the final predicted results based on the extracted features from the RNN-based feature extractor. As mentioned before, we adopt the KD ranking correlation coefficient as the evaluation metric for the predictor. However, typical regression loss functions, e.g., mean-absolute error (MAE, L1) loss and mean-squared error (MSE, L2) loss, are designed to optimize the linear correlation coefficient and are not suitable to optimize the ranking correlation coefficient. To improve the predictor performance, we construct the KD ranking loss function based on the ranking loss function in Gates [41] as the surrogate loss to optimize the KD ranking correlation. Our KD ranking loss function is named random pair margin loss (RPM loss). The RPM loss function goes as follows:

$$\text{Loss} = \sum_{i<j} \max\big(0, -\text{sign}\big(y_i - y_j\big) \cdot \big(s_i - s_j\big) + \text{margin}\big) \tag{8}$$

where y_i and s_i represent the ith ground-truth hardware performance and predicted hardware performance, respectively, margin represents the margin to control the loss function. The similar form to (6) enables the proposed RPM loss function to directly optimize the KD ranking correlation coefficient, realizing end-to-end gradient-based optimization.

As shown in Fig. 13, we compare the KD coefficients under L1 loss, L2 loss, and the proposed RPM loss functions for the area and energy consumption prediction. The experimental results show that our RPM loss function always reaches the best KD ranking correlation coefficient. Compared to the L1

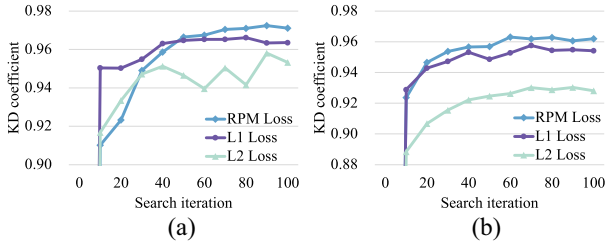


Fig. 13. KD coefficient comparison under different loss functions, i.e., L1 Loss, L2 Loss, and the proposed random pair margin (RPM) loss, for the prediction of (a) area and (b) energy consumption.

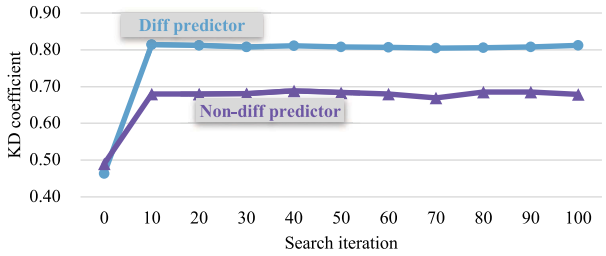


Fig. 14. KD coefficients of the prediction-for-difference predictor and the vanilla predictor on the same validation set of candidate designs.

loss function, the proposed RPM loss improves the KD coefficient by around 1% for both area and energy consumption prediction.

E. Differential Accuracy Predictor

Unlike other CMOS-based NN accelerators, PIM architectures face more severe computing errors caused by nonideal factors (e.g., device variation) and hardware quantization errors (e.g., ADC quantization). Thus, it is vital to predict the PIM-based NN accuracy precisely and efficiently during the co-exploration process. The PIM-based NN accuracy is heavily affected by the weights of NN models. However, the well-trained weights require a long training time, damaging the efficiency of the predictor. To tackle this problem, we propose an accuracy predictor which is independent of the NN weights. Different from vanilla predictor-based NAS methods [41] that predict the accuracy directly, *Gibbon* proposes to predict the relative accuracy loss brought by PIM architecture of a candidate design, as shown in Fig. 11. We use the one-shot accuracy [28] as the base accuracy, and the one-shot weights in the supernet are updated jointly in search iterations. Thanks to the design of "prediction for difference," the predictor only has to model the effects brought by the nonideal factors of PIM architectures, which is an easier problem than predicting the absolute accuracy.

Therefore, *Gibbon* manages to train a more accurate predictor with a small amount of evaluation results as training data. The experimental results in Fig. 14 show that the "prediction-for-difference" predictor can give out predictions with better KD ranking quality (∼10% improvement) using the same amount of training data.

F. Predictor Performance

To demonstrate the effectiveness of our predictor, we compare the evaluation time of the proposed predictor and the baseline PIM simulator, i.e., MNSIM 2.0. Fig. 15(a) shows that the baseline PIM simulator takes 549 s to evaluate one design candidate on average. Meanwhile, our predictor consumes only 7.59 s on average, reducing the evaluation time by 98.6%. And *Gibbon* uses the predictor to substitute for 95% of the simulation workload. As for the prediction error of our predictor, compared with the baseline PIM simulator, the relative prediction error is only 1.6% and 0.4% for the PIM-based NN accuracy and latency, respectively, as shown in Fig. 15(b).

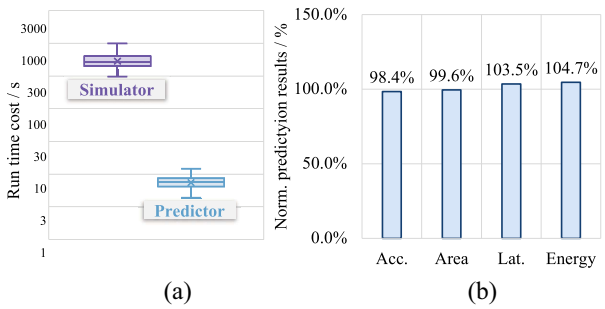


Fig. 15. (a) Evaluation time cost comparison of our predictor and the baseline PIM simulator. (b) Prediction results normalized to PIM simulator (Acc. and Lat. represent PIM-based NN accuracy and computing latency, respectively).

G. Predictor Construction

To obtain a precise predictor, we need to carry out two steps: 1) constructing the training set and 2) updating the weights of the predictor. The training set is constructed based on the PIM-oriented search space and the hardware performance simulator, e.g., MNSIM 2.0 [33] and NeuroSim [17]. First, we randomly sample 300 design candidates from the PIM-oriented search space. Afterward, we conduct performance simulation for these design candidates based on a time-consuming PIM simulator to acquire the PIM-based NN accuracy and hardware performance, e.g., area and EDP. Finally, 300 pairs of design candidates and their performance results constitute the training set of the predictor. The NN structure in each design candidate contains ten basic blocks, and the block structure affects the hardware performance directly. Therefore, the training set contains 3000 basic blocks. And our experimental results show that 300 pairs of design candidates and their performance are enough to train a precise performance predictor. The next step is updating the weights of the predictor based on the training set. We utilize gradient-based optimization methods to train the predictor iteratively. It should be noticed that once the predictor is obtained, we can apply it in *Gibbon* under different optimization targets.

VII. Experimental Results

A. Experiment Setup

Gibbon is developed based on aw_nas [43], an open-source NAS framework. Details of the searchable design hyperparameters are summarized in Table II. In this article, we use the most mature 1-bit and 2-bit memristors, and *Gibbon* also supports the search of other device precision. We adopt MNSIM 2.0 [33] as the baseline simulator to generate evaluation results for the predictor (other PIM simulators can also be used

TABLE II
NN STRUCTURE AND PIM-RELATED SEARCH SPACE

Block Number	1, 2
Output Channel Number	16, 32, 48, 64, 80, 96
Kernel Size	1, 2, 3
Group Number	1, 2, 4, 8, 16
Quantization Search Space	
Weight Bitwidth	5, 7, 9
Activation Bitwidth	5, 7, 9
Hardware Search Space	
Crossbar Size	32, 64, 128, 256
ADC Resolution	4, 6, 8, 10
DAC Resolution	1, 2
Memristor Precision	1, 2

TABLE III
PERFORMANCE (PIM-BASED NN ACCURACY, EDP, AREA, AND SEARCH TIME) COMPARISON OF DIFFERENT CO-EXPLORATION METHODS UNDER CIFAR-10 DATASET

Method	NN accuracy	EDP ($ms \times mJ$)	Area (mm^2)	Search time (h)
NACIM [13]	73.9%	1.55	17.17	59
UAE [14]	83.0%	–	–	154
NAS4RRAM [15]	84.4%	–	–	289
CARS [28] (acc opt.)	88.0%	11.03	227.73	72
Gibbon (edp opt.)	83.4%	**0.26**	142.77	**6**
Gibbon (area opt.)	78.5%	0.97	**13.68**	**6**
Gibbon (acc opt.)	**89.2%**	1.67	67.34	**6**

TABLE IV
PERFORMANCE (PIM-BASED NN ACCURACY, EDP, AREA) COMPARISON OF THE MANUALLY DESIGNED NNS AND THE PROPOSED CO-EXPLORATION FRAMEWORK UNDER CIFAR-10 DATASET

Method	NN accuracy	Energy (mJ)	Latency (ms)	EDP ($ms \times mJ$)	Area (mm^2)
AlexNet [44]	81.7%	0.38	0.99	0.38	103.99
VGG16 [18]	88.8%	2.68	6.43	17.22	499.57
VGGM[4] [45]	89.8%	5.41	11.26	60.87	316.05
ResNet18 [34]	89.7%	1.33	3.58	4.75	466.94
Gibbon (edp opt.)	83.4%	**0.13**	**1.99**	**0.26**	142.77
Gibbon (area opt.)	78.5%	0.16	6.07	0.97	**13.68**
Gibbon (acc opt.)	**89.2%**	0.49	3.44	1.67	67.34

TABLE V
PERFORMANCE (PIM-BASED NN ACCURACY, EDP, AND AREA) COMPARISON OF THE MANUALLY DESIGNED NNS AND THE PROPOSED CO-EXPLORATION FRAMEWORK UNDER CIFAR-100 DATASET

Method	NN accuracy	Energy (mJ)	Latency (ms)	EDP ($ms \times mJ$)	Area (mm^2)
AlexNet [44]	57.1%	0.38	1.00	0.38	103.99
VGG16 [18]	67.2%	2.68	6.44	17.25	499.57
VGGM [45]	70.4%	5.41	11.26	60.92	316.05
ResNet18 [34]	72.3%	1.33	3.59	4.76	466.94
Gibbon (edp opt.)	65.6%	**0.12**	**2.38**	**0.30**	144.93
Gibbon (area opt.)	61.3%	0.16	6.07	0.98	**19.01**
Gibbon (acc opt.)	**71.1%**	0.25	5.13	1.28	69.09

in *Gibbon*). The data of memristors, ADCs, and DACs in *Gibbon* refers to the default values provided by MNSIM 2.0. NACIM [13], UAE [14], and NAS4RRAM [15] are selected as the baseline NN model and PIM architecture co-exploration methods. We evaluate the proposed co-exploration framework on the CIFAR-10 and CIFAR-100 [8] datasets. We utilize MNSIM 2.0 to conduct performance simulation results for all experiments. The co-exploration process and performance simulation are performed in Intel Xeon E-5 2630 processors. We use NVIDIA® GeForce® RTX 2080 Ti devices to speed up the co-exploration.

B. Co-Exploration Results Comparison

Table III provides the co-exploration result comparison between *Gibbon* and other PIM-oriented NAS methods on the CIFAR-10 dataset. For a fair comparison, we evaluate the NN models discovered by NACIM [13] with MNSIM 2.0 to acquire the hardware performance. UAE [14] and NAS4RRAM [15] only provide PIM-based NN accuracy without giving other hardware performance. We also compare *Gibbon* with the vanilla CARS [28] co-exploration method without the proposed ESAPP and the predictor. We provide the co-exploration results of *Gibbon* under three different optimization targets (adjust the weight of each objective in the search reward): 1) Energy–Delay Product (EDP) optimization; 2) Area optimization; and 3) PIM-based NN accuracy optimization. All co-exploration time consumption is evaluated on the same Nvidia RTX 2080 Ti device.

Compared with other PIM-oriented NAS work, *Gibbon* can achieve 4.6%–15.3% PIM-based NN accuracy promotion in only six search hours. *Gibbon* improves the search efficiency by 9.8×–48.2×. Furthermore, compared with the vanilla CARS method, the proposed ESAPP and predictor show 12× co-exploration speed-up with better co-exploration performance. In terms of hardware performance, *Gibbon* with the EDP optimization achieves 5.96× EDP reduction with 9.5% accuracy improvement. As for the area, *Gibbon* with the area optimization realizes 1.25× area reduction.

We compare the co-exploration results of *Gibbon* with the manually designed NNs, e.g., AlexNet [44], two kinds of VGG [18], [45], and ResNet [34], on both the CIFAR-10 and CIFAR-100 datasets. As shown in Tables IV and V, compared with manually designed NNs, *Gibbon* with the EDP optimization reduces EDP by 1.46×–234.1×, and *Gibbon* with the area optimization reduces the area consumption by 5.47×–36.5×. As for the PIM-based NN accuracy optimization, *Gibbon* can achieve 14.1× EDP reduction and 5.12× area reduction on average with comparable PIM-based NN accuracy. The slight accuracy loss comes from that we limit the output channel numbers of convolutional operations to 96 for lower hardware resource cost. Meanwhile, those of manually designed NNs can rise to 512 or 1024 (5.33×–10.6×).

Fig. 16 demonstrates the PIM-based NN accuracy and EDP performance of *Gibbon* with different optimization targets and NACIM. The performance results of *Gibbon* w/o ESAPP and w/ESAPP are provided under the same search time. The superiority of *Gibbon* w/ESAPP shows that ESAPP can find better results in the same search time. Compared with the NACIM method, *Gibbon* with EDP optimization can find the better Pareto frontier with lower EDP and higher PIM-based NN accuracy.

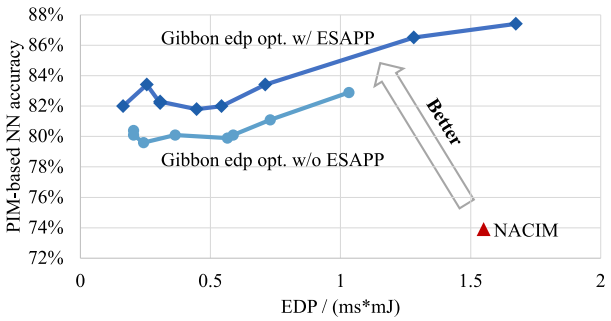


Fig. 16. Co-exploration results (PIM-based NN accuracy and EDP) of NACIM, *Gibbon* with EDP optimization w/ and w/o ESAPP.

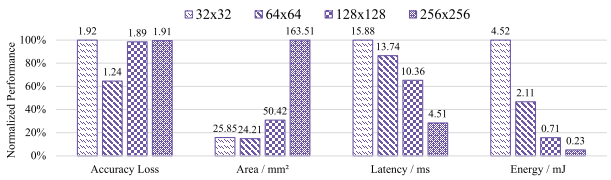


Fig. 17. PIM-based NN accuracy loss and hardware performance comparison among different crossbar sizes.

VIII. Insights Provided by Gibbon

By inspecting the NN models and PIM architectures discovered by *Gibbon*, we find some interesting observations as follows, hoping to provide some design suggestions for the co-design of the NN models and PIM architectures in the future.

Insight 1: Gibbon with EDP optimization finds that most convolutional operations tend to select even-sized kernels. When other design hyperparameters stay the same, design candidates with 2×2 kernels can decrease $\sim$84% EDP and $\sim$35% area compared with those with 3×3 kernels.

Insight 2: Gibbon with PIM-based NN accuracy optimization finds that convolutional operations in the head and tail blocks tend to have larger output channels, e.g., 64, while those in the middle blocks have smaller ones, e.g., 16.

Insight 3: Gibbon with PIM-based NN accuracy optimization finds that the deeper convolutional operations tend to choose high quantization bitwidth of weights. At the same time, the shallower convolutional operations prefer low weights precision. For the activation quantization, both the head and tail convolutional operations prefer high bitwidth, while the middle tends to choose low bitwidth for lower hardware resource cost.

Insight 4: On the CIFAR-10 and CIFAR-100 dataset, the PIM-based NN accuracy of 8-bit ADCs is close to that of 10-bit ADCs (i.e., 0.2% accuracy difference). Nevertheless, 6-bit ADCs lead to non-negligible accuracy loss.

Insight 5: Gibbon with EDP optimization finds that the number of output channels of blocks in the early stages impacts the total latency significantly. *Gibbon* with the latency optimization tends to assign smaller output channels in the early stages.

Insight 6: As shown in Fig. 17, we compare the average accuracy loss and hardware performance under different crossbar sizes based on the predictor. 64×64 crossbars realize the lowest accuracy loss, which is $\sim$36% lower than other crossbar sizes. *Gibbon* with area optimization also finds that crossbars in size of 64×64 realize the smallest area. The energy and latency optimal PIM design tends to choose a large crossbar size (e.g., 256×256), which can reduce the amount of analog-to-digital conversions. Compared with crossbars in size of 128×128, PIM architectures with 256×256 crossbars can reduce energy consumption by around 68%.

IX. Conclusion

In this article, we propose *Gibbon* to efficiently co-explore the NN model and PIM architecture. Compared with existing PIM-oriented NAS work, *Gibbon* leverages ESAPP and an RNN-based predictor to improve search efficiency. Experimental results show that *Gibbon* can achieve $9.8\times$–$48.2\times$ co-exploration speedup with up to 15.3% PIM-based NN accuracy improvement and $5.96\times$ EDP reduction compared with existing work.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.

[2] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–15.

[3] P. Chi et al., "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 27–39, 2016.

[4] L. Song et al., "Pipelayer: A pipelined ReRAM-based accelerator for deep learning," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2017, pp. 541–552.

[5] M. Cheng et al., "TIME: A training-in-memory architecture for RRAM-based deep neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 834–847, May 2019.

[6] Z. Zhu et al., "A configurable multi-precision CNN computing framework based on single bit RRAM," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[7] T.-H. Yang et al., "Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proc. 46th Int. Symp. Comput. Archit.*, 2019, pp. 236–249.

[8] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Dept. Comput. Sci., Univ. Toronto, Toronto, ON, Canada, Rep. TR-2009, 2009.

[9] C. Tang, W. Sun, W. Wang, and Y. Liu, "Dynamic CNN accelerator supporting efficient filter generator with kernel enhancement and online channel pruning," in *Proc. 27th Asia–South Pacific Design Autom. Conf. (ASP-DAC)*, 2022, pp. 436–441.

[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[11] 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.

[12] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 1–9.

[13] W. Jiang et al., "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 595–605, Apr. 2021.

[14] Z. Yan, D.-C. Juan, X. S. Hu, and Y. Shi, "Uncertainty modeling of emerging device based computing-in-memory neural accelerators with application to neural architecture search," in *Proc. 26th Asia–South Pacific Design Autom. Conf. (ASP-DAC)*, 2021, pp. 859–864.

[15] Z. Yuan et al., "NAS4RRAM: Neural network architecture search for inference on RRAM-based accelerators," *Sci. China Inf. Sci.*, vol. 64, no. 6, pp. 1–11, 2021.

[16] L. Xia et al., "MNSIM: Simulation platform for memristor-based neuromorphic computing system," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1009–1022, May 2018.

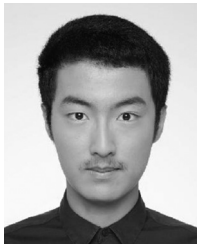
[17] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "DNN+NeuroSim V2.0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 11, pp. 2306–2319, Nov. 2021.
[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[20] H. Akinaga and H. Shima, "Resistive random access memory (ReRAM) based on metal oxides," *Proc. IEEE*, vol. 98, no. 12, pp. 2237–2251, Dec. 2010.
[21] F. Xia, D.-J. Jiang, J. Xiong, and N.-H. Sun, "A survey of phase change memory systems," *J. Comput. Sci. Technol.*, vol. 30, no. 1, pp. 121–144, 2015.
[22] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, "Computing in memory with spin-transfer torque magnetic RAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 3, pp. 470–483, Mar. 2018.
[23] A. Shafiee et al., "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.
[24] P. Ren et al., "A comprehensive survey of neural architecture search: Challenges and solutions," *ACM Comput. Surveys*, vol. 54, no. 4, pp. 1–34, 2021.
[25] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4095–4104.
[26] 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.*, vol. 34, no. 2, pp. 550–570, Feb. 2023.
[27] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 4780–4789.
[28] Z. Yang et al., "CARS: Continuous evolution for efficient neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1829–1838.
[29] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.
[30] H. Benmeziane, K. El Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, "Hardware-aware neural architecture search: Survey and taxonomy," in *Proc. IJCAI*, 2021, pp. 4322–4329.
[31] Y. Lin, M. Yang, and S. Han, "NAAS: Neural accelerator architecture search," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 1051–1056.
[32] L. Yang et al., "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.
[33] Z. Zhu et al., "MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems," in *Proc. Great Lakes Symp. VLSI*, 2020, pp. 83–88.
[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[35] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.
[36] Z. Zhu et al., "Mixed size crossbar based RRAM CNN accelerator with overlapped mapping method," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2018, pp. 1–8.
[37] S. Wu, G. Wang, P. Tang, F. Chen, and L. Shi, "Convolution with even-sized kernels and symmetric padding," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–12.
[38] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.
[39] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[40] Y. Hu et al., "Angle-based search space shrinking for neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 119–134.
[41] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang, "A generic graph-based neural architecture encoding scheme for predictor-based NAS," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 189–204.
[42] H. Robbins and S. Monro, "A stochastic approximation method," in *Proc. Ann. Math. Stat.*, 1951, pp. 400–407.
[43] X. Ning et al., "aw_nas: A modularized and extensible NAS framework," 2020, *arXiv:2012.10388*.
[44] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017.
[45] Y. Cai, T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang, "Low bit-width convolutional neural network on RRAM," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1414–1427, Jul. 2019.

Hanbo Sun received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Electronic Engineering.

His research interests focus on neural architecture search for processing-in-memory (PIM) systems and enhancing energy efficiency for PIM systems.

Zhenhua Zhu received the B.S. degree from the Electronic Engineering Department, Tsinghua University, Beijing, China, in 2018, where he is currently pursuing the Ph.D. degree.

His research mainly focuses on memristor, computer architecture, and processing-in-memory.



Chenyu Wang is currently pursuing the B.S. degree with the Department of Electronic Engineering, Tsinghua University, Beijing, China.

His research interests focus on processing-in-memory.

Xuefei Ning received the B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 2016 and 2021, respectively.

His research focuses on efficient deep-learning algorithm design and neural architecture search.

Guohao Dai (Member, IEEE) received the B.S. and Ph.D. (with Hons.) degrees from Tsinghua University, Beijing, China, in 2014 and 2019, respectively.

He is currently an Associate Professor Shanghai Jiao Tong University, Shanghai, China. His research mainly focuses on large-scale sparse graph computing, heterogeneous hardware computing, and emerging hardware architecture.

Dr. Dai has received the Best Paper Award in ASP-DAC 2019 and the Best Paper Nomination in DAC 2022 and DATE 2018. He is the winner of the NeurIPS Billion-Scale Approximate Nearest Neighbor Search Challenge in 2021 and the recipient of the Outstanding Ph.D. Dissertation Award of Tsinghua University in 2019. He currently serves as a PI/Co-PI for several projects with a personal share of over RMB 6 million.

Huazhong Yang (Fellow, IEEE) received the B.S. degree in microelectronics and the M.S. and Ph.D. degrees in electronic engineering from Tsinghua University, Beijing, China, in 1989, 1993 and 1998, respectively.

In 1993, he joined the Department of Electronic Engineering, Tsinghua University, where he has been a Professor since 1998. He has authored and coauthored over 500 technical papers, seven books, and over 180 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain-inspired computing.

Prof. Yang was awarded the Distinguished Young Researcher by NSFC in 2000, the Cheung Kong Scholar by the Chinese Ministry of Education (CME) in 2012, the Science and Technology Award First Prize by China Highway and Transportation Society in 2016, and the Technological Invention Award First Prize by CME in 2019. He has been in charge of several projects, including projects sponsored by the National Science and Technology Major Project, 863 Program, NSFC, and several international research projects. He has also served as the Chair for Northern China ACM SIGDA Chapter Science 2014, the General Co-Chair of ASP-DAC'20, a Navigating Committee Member of AsianHOST'18, and a TPC Member for ASP-DAC'05, APCCAS'06, ICCCAS'07, ASQED'09, and ICGCS'10.

Yu Wang (Fellow, IEEE) received the B.S. and Ph.D. (with Hons.) degrees from Tsinghua University, Beijing, China, in 2002 and 2007, respectively.

He is currently a Tenured Professor with the Department of Electronic Engineering, Tsinghua University. He has authored and coauthored more than 350 papers in refereed journals and conferences. His research interests include brain-inspired computing, parallel circuit analysis, application-specific acceleration, power/reliability-aware circuit, and system design methodology.

Dr. Wang has received the Best Paper Award in ASP-DAC 2019, FPGA 2017, NVMSA 2017, and ISVLSI 2012, the Best Poster Award in HEART 2012, and 11 Best Paper Nominations (DAC22, ICT18, DATE18, DAC17, ASPDAC16, ASPDAC14, ASPDAC12, 2 in ASPDAC10, ISLPED09, and CODES09). He is a recipient of the Alexander von Humboldt Fellowship in 2019, the DAC Under-40 Innovators Award in 2018, and the IBM X10 Faculty Award in 2010. He served as a TPC Chair for ICFPT 2019 and 2011, ISVLSI 2018, a Finance Chair for ISLPED from 2012 to 2016, a Track Chair for DATE from 2017 to 2019 and GLSVLSI 2018, and a Program Committee Member for leading conferences in these areas, including top EDA conferences, such as DAC, DATE, ICCAD, and ASP-DAC, and top FPGA conferences, such as FPGA and FPT. He currently serves as an Associate Editor for the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, the *ACM Transactions on Design Automation of Electronic Systems*, and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.