

CrossNAS: A Cross-Layer Neural Architecture Search Framework for PIM Systems

Md Hasibul Amin

University of South Carolina
Columbia, SC, USA
ma77@email.sc.edu

Jason D. Bakos

University of South Carolina
Columbia, SC, USA
jbakos@cse.sc.edu

Mohammadreza Mohammadi

University of South Carolina
Columbia, SC, USA
mohammam@email.sc.edu

Ramtin Zand

University of South Carolina
Columbia, SC, USA
ramtin@cse.sc.edu

Abstract

In this paper, we propose the CrossNAS framework, an automated approach for exploring a vast, multidimensional search space that spans various design abstraction layers—circuits, architecture, and systems—to optimize the deployment of machine learning workloads on analog processing-in-memory (PIM) systems. CrossNAS leverages the single-path one-shot weight-sharing strategy combined with the evolutionary search for the first time in the context of PIM system mapping and optimization. CrossNAS sets a new benchmark for PIM neural architecture search (NAS), outperforming previous methods in both accuracy and energy efficiency while maintaining comparable or shorter search times.

Keywords

processing-in-memory, neural architecture search, weight-sharing, evolutionary algorithm, mixed-precision quantization

1 Introduction

Processing-in-memory (PIM) architectures have emerged as promising alternatives to conventional von Neumann-based machine learning (ML) hardware [15]. These architectures exploit features such as massive parallelism, analog computation, and the ability to perform computations directly where data is stored, leading to significant performance improvements [4, 8, 10, 24]. The foundation of most PIM architectures is based on memristive crossbar arrays, which utilize resistive memory technologies such as resistive random-access memory (RRAM) and magnetoresistive random-access memory (MRAM) [32]. These arrays enable matrix-vector multiplication (MVM) in the analog domain using basic circuit laws [9, 13].

Despite the progress in PIM architectures, previous research indicates that deploying pre-trained ML models, designed and optimized for digital von Neumann architectures, does not consistently achieve comparable performance on analog PIM architectures [2]. This is caused by several factors, including the limited numerical precision of memristive devices [20], and circuit imperfections such as interconnect parasitics [1], and device variations [16].

An alternative strategy for mapping and deploying ML models to PIM architecture includes the automated tuning of neural architecture parameters alongside PIM circuit and device-level hyperparameters to achieve specific design objectives [3]. NACIM [14] and UAE [30] provide such exploration frameworks based on

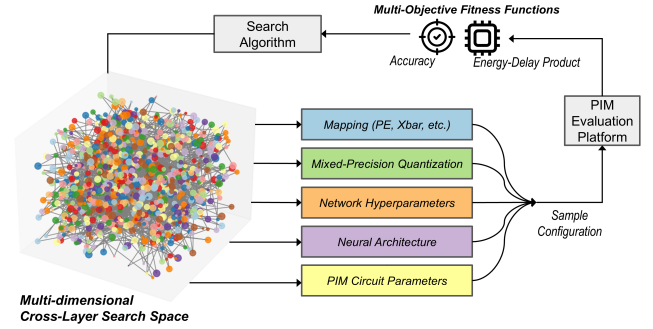


Figure 1: The CrossNAS method explores a multi-dimensional search space across multiple design abstraction layers—circuit, architecture, and system levels—using a custom multi-objective search and optimization process.

reinforcement learning and LSTM controller, respectively, but they explore only VGG-like models, which limits their performance. NAS4RRAM [31] also provides a similar framework for ResNet-like architectures. NACIM, UAE and NAS4RRAM need to train the sampled architectures from scratch to estimate the accuracy which increases the search time. NAX [21] trains an over-parameterized network for both architecture and hardware optimization but their architecture search space only contains filter sizes for each layer, lacking block types or channel numbers. Gibbon [28] uses a recurrent neural network (RNN)-based predictor for estimation of accuracy and hardware performance, but it only includes ResNet-like blocks in their architecture. AnalogNAS [6] trains a surrogate model which predicts the behavior of searched architectures but they mostly focus on ResNet-like architectures and PIM configuration parameters are not included in the search space.

In this work, we present the CrossNAS framework, an automated approach for exploring a vast multidimensional search space across PIM design abstraction layers, including circuit, architecture, and system levels, as shown in Fig. 1. We propose a weight-sharing-based neural architecture search (NAS) approach to explore the cross-layer search space, encompassing diverse neural building blocks, convolutional channel configurations, layer-specific quantization, and PIM circuit-level hyperparameters.

The main challenge of the weight-sharing approaches is to overcome the weight coupling among different architectures within

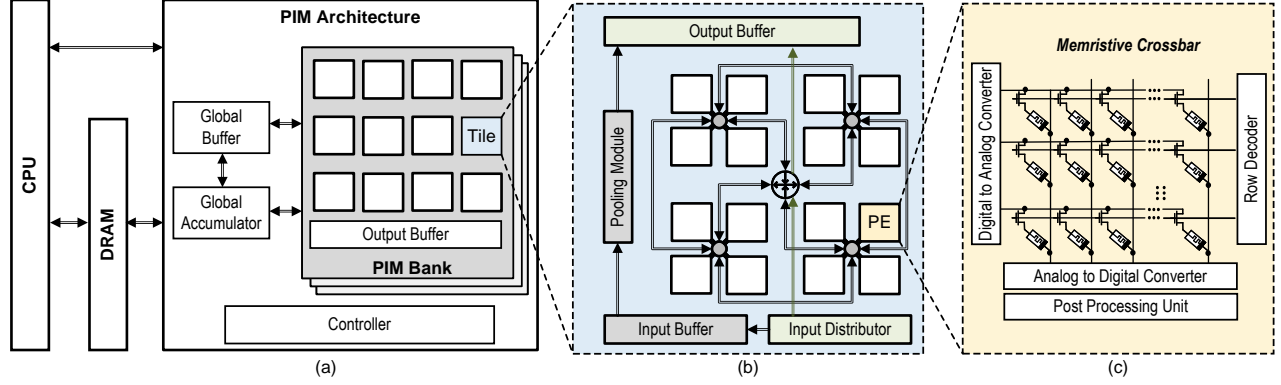


Figure 2: (a) The analog PIM architecture with multiple banks including several interconnected tiles. (b) The PIM tile consists of a network of processing elements (PEs). (c) The PEs include memristive crossbars and signal conversion units.

the supernet, which raises concerns about the effectiveness of inherited weights for a specific architecture. Various weight-sharing approaches have been previously explored for neural architecture search [5, 7, 11, 22, 26, 29, 33]. Many weight-sharing methods employ continuous relaxation to parameterize the search space, where architecture distribution parameters are jointly optimized during supernet training using gradient-based techniques [22, 29]. After optimization, the best architecture is sampled from the distribution. However, the joint optimization method entangles architecture parameters with supernet weights, and the greedy nature of gradient-based methods introduces bias, potentially misleading the architecture search. The one-shot paradigm [5, 7] alleviates this issue by decoupling the architecture search from supernet training. It defines the supernet and performs weight inheritance similarly, without architecture relaxation. However, the issue of weight coupling among different architectures still persists. To overcome this issue, single-path [11, 26, 33] NAS has been proposed, where all architectures are optimized simultaneously, through uniform sampling during supernet training.

The CrossNAS framework utilizes the single-path one-shot [11] weight-sharing strategy to find the optimum neural model and corresponding PIM parameters. The process involves training an overparameterized supernet based on neural search space, followed by using an evolutionary algorithm to select optimal subnets based on multi-objective fitness functions. The contributions of our work can be summarized as follows:

- Creating a multi-dimensional cross-layer search space including different architectures, channel counts, layer-specific weight and activation bit widths, as well as PIM parameters such as crossbar size and the precision of analog-to-digital (ADC) and digital-to-analog (DAC) converters. We introduce new parameters to the PIM search space, allowing for the selection of different neural building blocks for each layer of the model.
- Adapting the single-path one-shot weight-sharing strategy along with the evolutionary search for the first time in the context of mapping and optimization of PIM systems.
- Establishing a new benchmark for PIM neural architecture search that surpasses prior works in terms of accuracy and energy efficiency.

2 PIM Architecture

Figure 2 illustrates the structure of the analog PIM architecture utilized in this work [34]. This architecture features multiple PIM banks, a global buffer, a global accumulator, and a PIM controller, as shown in Fig. 2 (a). The PIM controller oversees data transfer between DRAM and PIM banks and receives status updates from the CPU. The global buffer and global accumulator support elementwise summation, which is crucial for implementing skip connections in ML models such as ResNet [12].

Each PIM bank is structured into arrays of tiles interconnected via a network-on-chip (NoC). Within each tile are processing elements (PEs), a pooling module, and input/output buffers, as depicted in Fig. 2 (b). The PEs include memristive crossbars and peripheral circuits such as ADCs, DACs, and a post-processing unit for handling nonlinear vector operations. The memristive crossbars execute MVM operations in the analog domain by leveraging basic circuit principles: Ohm’s law is used for multiplication ($I = GV$), while Kirchhoff’s law enables accumulation via charge conservation [2, 13]. The weight kernels are expanded into a vector and loaded onto crossbar columns by adjusting the conductance (G) of memristive devices, while the input feature maps are applied as input voltages (V) to the crossbar. In this work, we use MNSIM 2.0 [34] to emulate PIM circuit behavior and map ML models onto PIM hardware, allowing us to measure model accuracy, as well as PIM energy consumption and latency.

3 The Proposed CrossNAS Methodology

Figure 3 provides a high-level description of the proposed CrossNAS framework which includes four main steps. ① Train an overparameterized supernet comprising all possible architectures in the search space using a single-path one-shot weight-sharing strategy. ② Use an evolutionary algorithm to search subnet architectures and evaluate them based on a multi-objective fitness function (FF) comprising accuracy and performance metrics. We use corresponding inherited weights from the supernet to approximate accuracy for the subnets, so no retraining is required. We adopt a behavior-level PIM simulator MNSIM 2.0 [34] for fast calculation of the hardware performance metrics. ③ The single-path one-shot weight-sharing strategy is utilized to train a mixed-precision quantization supernet based on the selected neural network (NN) architecture comprising

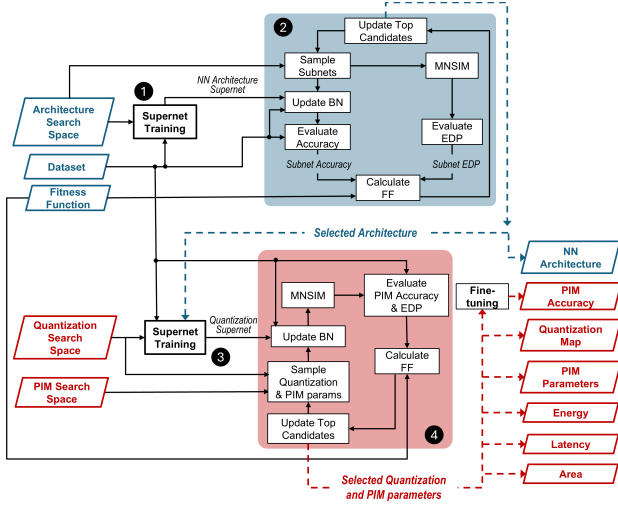


Figure 3: Optimization flow for the CrossNAS framework.

all possible quantization scenarios in the quantization search space. ④ Using an evolutionary algorithm, we search the quantization space and PIM parameters to maximize FF. To calculate the FF, supernet weights are sent to MNSIM to measure the PIM-based NN accuracy and hardware performance metrics for the candidate quantization map and PIM parameters.

The single-path one-shot NAS method is described in section 3.1. The supernet training and subnet architecture search for NN architectures (steps ① and ②) are described in section 3.2. Finally, section 3.3 describes the mixed-precision quantization supernet training along with the evolutionary search of quantization space and PIM hyperparameters (steps ③ and ④).

3.1 Single-path One-shot NAS

According to the single-path one-shot NAS approach [11], the supernet weight training and the subnet architecture search are performed in two separate sequential steps. The supernet is an overparameterized network which contains the trained weights for all possible architectures in the search space. The subnet is a subset of the supernet which is evaluated using its inherited weights from the supernet. For the supernet training, our goal is to minimize the training loss for all possible architecture choices in the search space as shown in equation (1),

$$W_{SUP} = \underset{W}{\operatorname{argmin}} \mathcal{L}_{train}(\mathcal{N}(A, W)) \quad (1)$$

where A is the architecture search space, $\mathcal{N}(A, W)$ represents the set of all subnet architectures, \mathcal{L}_{train} is the loss function. To ensure that the accuracy of a sampled architecture a on the test dataset using inherited weights $W_{SUP}(a)$ is highly predictive of the accuracy of fully-trained architecture a , we simultaneously optimize the weights for all architectures in the search space through uniform sampling. So, equation (1) can be rewritten as,

$$W_{SUP} = \underset{W}{\operatorname{argmin}} \mathbb{E}_{a \sim \Gamma(A)} [\mathcal{L}_{train}(\mathcal{N}(a, W_{SUP}(a)))] \quad (2)$$

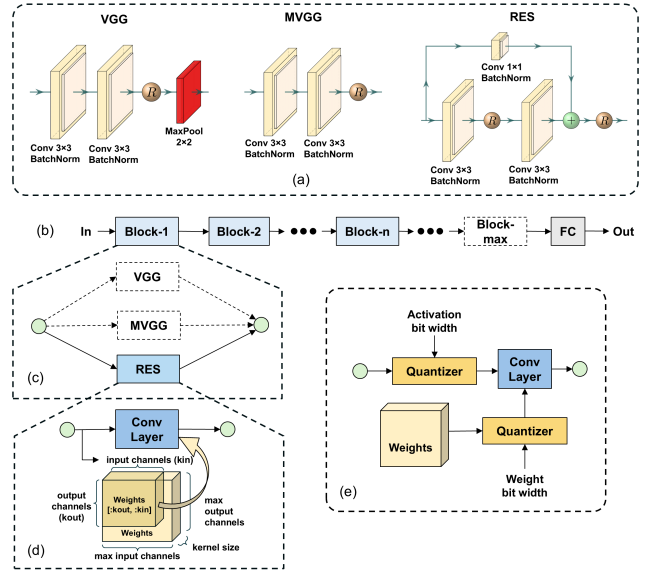


Figure 4: Hierarchical sampling of subnet from supernet. (a) building blocks, (b) NN architecture buildup by depth sampling, (c) block sampling, (d) output channel sampling via slicing of the weight matrix, and (e) layerwise weight and activation quantization

where $\Gamma(A)$ is a prior distribution of $a \in A$. Thus, in each optimization step, we randomly sample one architecture a and only its corresponding weight $W_{SUP}(a)$ is activated and trained. Therefore, By the end of training, the supernet acts as an approximate model, representing the weights of fully-trained subnet architectures.

Once the supernet is trained, we sample subnets from the trained supernet architecture and evaluate them based on the desired FF. During the subnet search, each of the sampled architecture inherits weights from the supernet W_{SUP} as $W_{SUP}(a)$. The best architecture is selected as,

$$a^* = \underset{a \in A}{\operatorname{argmax}} FF_{test} \mathcal{N}(a, W_{SUP}(a)) \quad (3)$$

Since each fitness function (FF_{test}) utilized in this work includes accuracy as a key metric, we assess accuracy through inference using weights inherited from the supernet. With access to these pre-trained supernet weights, our search process becomes highly efficient, eliminating the need for retraining. To search for optimized subnets based on specific FFs, we apply an evolutionary algorithm as described in the following.

3.2 Neural Network (NN) Architecture Search

As shown in Figure 4, we build an overparameterized supernet containing the weights for all possible architecture choices in the search space. As building blocks for our network configuration, we design three different convolutional choice blocks based on the well-known VGG [25] and ResNet [12] architectures, as shown in Fig. 4 (a). The three building blocks are: (1) the VGG block with two 3×3 convolution layers with a stride of 1 followed by a 2×2 max pooling layer, (2) the modified VGG block (MVGG) which is similar to VGG without the sub-sampling layer, and (3) the RES

Table 1: Network Configuration Settings

Architecture Search Space	
No. of blocks in the network (n)	1-8
The block type ($Block$)	VGG, MVGG, RES
No. of out. channels in each block (K)	32, 64, 128
Quantization Search Space	
Weight bit width (WB)	5, 7, 9
Activation bit width (AB)	5, 7, 9
PIM Search Space	
Crossbar size ($Xbar$)	32, 64, 128, 256
ADC resolution (ADC)	4, 6, 8, 10
DAC resolution (DAC)	1, 2

block with two 3×3 convolution layers and a residual connection with a 1×1 convolution block. The ReLU activation function is used as the nonlinear unit in all of the building blocks. The full search space is shown in Table 1.

As shown in Fig. 4 (b), the supernet is configured with the maximum depth (D_{max}) and includes the highest possible number of sequential convolutional (conv) blocks within the search space, followed by fully connected layers as classification heads. Each conv block in the supernet includes all three types of VGG, MVGG, and RES blocks connected in parallel paths (Fig. 4 (c)). The size of the conv layers weight matrices in the supernet are set according to maximum input (k_{in}) and output (k_{out}) channel sizes possible (Fig. 4 (d)). At each training step, the algorithm randomly selects a depth of n blocks, one of the VGG, MVGG, or RES paths in each block, and output channel count (k_{out}) for each block. Based on this selection, the first n conv blocks and their corresponding paths are activated, and the remaining $D_{max} - n$ blocks are skipped. The weight matrices of active conv layers and paths are then sliced according to the selected input and output channels, updating only the relevant kernel portions in that training step.

During the search phase, we employ an evolutionary algorithm to identify top subnet candidates based on a multi-objective FF. We introduce a PIM-oriented FF, defined as a weighted sum of accuracy and the normalized energy-delay product (EDP), as shown in equation (4).

$$FF = w_{acc} \times accuracy - (1 - w_{acc}) \times EDP_{norm} \quad (4)$$

where w_{acc} can be tuned to any number between 0 and 1 to adjust the balance between accuracy and EDP.

We utilize the MNSIM 2.0 [34] simulator to determine the energy and latency of the selected subnet model’s PIM implementation. The w_{acc} is set based on user preference. Since the accuracy of each subnet—obtained using the inherited supernet weights—offers a reliable estimate for comparing subnet architectures, we use this approximate accuracy rather than performing the more time-intensive PIM-specific accuracy measurement in this phase. Because the supernet’s batch normalization (BN) statistics do not apply to candidate subnets, we retrain only the BN layers on the training set before evaluating subnet accuracy via inference. This BN retraining takes just a few seconds, adding minimal computational or time cost to the process.

We employ Algorithm 1 to identify the best-performing candidate subnet architecture from the search space. First, we initialize a candidate set P_i with P random subnets, which will represent the

Algorithm 1: Evolutionary Subnet Search

Input: supernet weights (W_{SUP}), training data ($train$), test data ($test$), population (P), maximum cycle (cyc), top k candidates ($topK$), mutation value (n), crossover value (m), mutation probability ($prob$)

Output: The NN architecture with the best value of corresponding fitness function

```

1 Initialize:  $P = 50$ ,  $P_0$ =random  $P$  candidates,  $topK = \phi$ ,
    $n = P/2$ ,  $m = P/2$ ,  $prob=0.1$ 
2 for  $i = 1$  to  $cyc$  do
3    $FF_{i-1} \leftarrow \text{Calculate\_FF}(W_{SUP}, train, test, P_{i-1})$ 
4    $topK \leftarrow \text{Update\_topK}(topK, P_{i-1}, FF_{i-1})$ 
5    $P_{crossover} \leftarrow \text{Crossover}(topK, n)$ 
6    $P_{mutation} \leftarrow \text{Mutation}(topK, m, prob)$ 
7    $P_i \leftarrow P_{crossover} \cup P_{mutation}$ 
8 end
9 Return the NN architecture with the highest value in  $topK$ 

```

new population in each search cycle. Additionally, we maintain a set $topK$ to store the top k subnet candidates with the highest fitness. After calculating the FF for candidates in P_i , we update $topK$ in descending order based on fitness scores. Based on the $topK$ candidates and a predefined mutation probability $prob$, we generate m crossover and n mutation candidates, with the newly generated candidates forming the updated P_i . This process is repeated for a fixed number of cycles (cyc). Ultimately, the candidate with the highest fitness value is selected as the optimal architecture.

3.3 Quantization and PIM Configuration Search

We apply a single-path one-shot weight-sharing method to train a mixed-precision quantization supernet, followed by a search for optimal mixed-precision quantization and PIM configurations using an evolutionary algorithm. This search is performed on the optimal model architecture identified in the preceding NN architecture search step, which defines the supernet architecture for this stage. During training, weight and activation bit widths are randomly sampled from the search space at each step, and these sampled bit precisions are dynamically applied to the model, as shown in Fig. 4 (e). As a result, the supernet model approximates the fully trained mixed-precision quantized weights, allowing for efficient representation of all possible mixed-precision subnet configurations.

We use a non-uniform quantization scheme [27] to quantize the weights and activations, as shown in the below equation.

$$x_q = \text{clip} \left(\text{round} \left(\theta \times \frac{x_{in}}{\alpha} \right), -\theta, \theta \right) \times \frac{\alpha}{\theta} \quad (5)$$

where α is the scaling parameter and $[-\theta, \theta]$ is the integer range. θ is calculated as $2^{q-1} - 1$, where q is the quantization bit width. For weights, α is set to the maximum absolute value in the weight tensor. The activations have a broader range, which may include individual extreme values. So, for a fair reflection of the overall data distribution, we use $|\mu| + 3|\sigma|$ as the scaling parameter for activations, where μ and σ are the mean and standard deviation of the activation data, respectively. The scaling parameter α can

be different in different mini-batches, which may cause jitter and non-convergence in training [27]. Therefore, each time a new α is generated, it is combined with the α from the previous mini-batch using a weighted sum, as shown in the equation (6).

$$\alpha = m\alpha + (1 - m) \times (|\mu(x_{in})| + 3|\sigma(x_{in})|) \quad (6)$$

To train a quantized model effectively, we begin by training a floating-point model and then fine-tune its weights to develop a mixed-precision quantization supernet. A fitness function and evolutionary algorithm, similar to those used in the NN architecture search step, are applied to explore various mixed-precision quantization mappings and PIM configurations. We update the batch normalization (BN) weights for the quantization subnet candidate before evaluating PIM-based NN accuracy in MNSIM. A lower mutation probability is assigned to the quantization map and a higher mutation probability to the PIM configuration parameters so that the top quantization candidates are cross-checked with different PIM configurations. Once the best subnet candidate is identified, we fine-tune the model for one last time to achieve optimal PIM accuracy.

4 Results and Discussions

4.1 Experimental Setup

The proposed CrossNAS framework is compatible with any PIM simulator that provides accuracy, energy, and delay metrics. For our experiments, we use MNSIM 2.0 [34] as the baseline simulator. The PIM accelerator is configured with a 64×64 tile arrangement, with each tile containing 2×2 PE arrays, and uses 1-bit memristor devices as the basic crossbar elements. We evaluate the performance of CrossNAS against previous exploration methods, including NACIM [14], UAE [30], NAS4RRAM [31], and Gibbon [28], on the CIFAR-10 and CIFAR-100 [18] datasets. CIFAR-10 is selected because it serves as a common benchmark used in all previous related studies, allowing consistent comparisons. All experiments are conducted on a single NVIDIA® GeForce® RTX™ 2080 Ti GPU paired with an Intel® Core™ i9-9820X CPU at 3.30GHz.

4.2 Training and Search Details

During the neural network architecture search phase, we train the supernet using stochastic gradient descent (SGD) optimizer [23] for 1000 epochs, starting with a learning rate of 0.1 and a batch size of 128. A learning rate scheduler is applied, reducing the learning rate by a factor of 5 every 250 epochs. The evolutionary algorithm runs for 10 cycles, evaluating 50 new candidates per cycle, consisting of 25 mutation candidates and 25 crossover candidates.

In the mixed-precision quantization and PIM configuration search phase, training begins with a floating-point (FP) model to achieve high accuracy. An SGD optimizer is used with an initial learning rate of 0.1, training for 200 epochs, with a step learning rate scheduler reducing the rate by a factor of 5 at epochs 60, 120, and 160. The FP weights are then used to retrain the model with variable mixed-precision quantization, employing the Adam optimizer [17] for quantization-aware training (QAT). Due to the sensitivity of QAT to learning rate, a low initial learning rate of 0.0008 is set, which is further divided by 5 every 40 epochs using a step learning rate scheduler.

Table 2: Performance comparison against different PIM-oriented NAS methods on CIFAR-10 dataset.

Method	PIM Accuracy	Latency (ms)	EDP ($mJ \times ms$)	Search time (h)
NACIM [14]	73.9%	-	1.55	59
UAE [30]	83%	-	-	154
NAS4RRAM [31]	84.4%	-	-	289
Gibbon [28] (acc opt.)	89.2%	3.44	1.67	6
Gibbon [28] (edp opt.)	83.4%	1.99	0.26	6
CrossNAS ($w_{acc}=0.99$)	91.27%	1.35	0.28	6
CrossNAS ($w_{acc}=0.8$)	88.09%	0.577	0.073	5

Table 3: Performance comparison against various well-known deep learning models on CIFAR-10 Dataset.

Method	PIM Accuracy	Energy (mJ)	Latency (ms)	EDP ($mJ \times ms$)	Area (mm^2)
AlexNet [19]	81.7%	0.38	0.99	0.38	103.99
VGG16 [25]	88.8%	2.68	6.43	17.22	499.57
ResNet18 [12]	89.7%	1.33	3.58	4.75	466.94
CrossNAS ($w_{acc}=0.99$)	91.27%	0.21	1.35	0.28	306.64
CrossNAS ($w_{acc}=0.8$)	88.09%	0.127	0.577	0.073	106.3

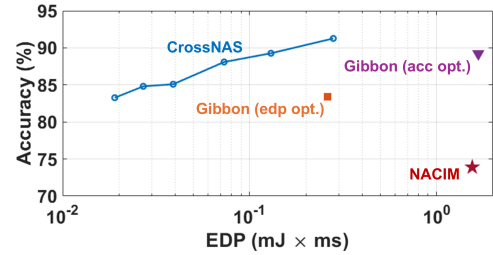


Figure 5: CrossNAS search results on CIFAR-10 dataset compared to previous PIM-oriented NAS methods.

4.3 Performance Comparison Results on CIFAR-10 Dataset

Table 2 presents a comparison of our results with previous PIM-oriented NAS frameworks on the CIFAR-10 dataset.

To obtain a model optimized for high accuracy, we set the w_{acc} to 0.99 in the FF. The model selected by CrossNAS at the end of optimization, shown in Fig. 6 (a), achieves an accuracy improvement of 2.07% to 17.37% over previous frameworks. The search process takes approximately six hours, comparable to Gibbon and up to 48.6× faster than other NAS methods. This model also achieves a 6× reduction in energy-delay-product (EDP) compared to the model selected by Gibbon under accuracy optimization.

Next, we optimize the model for EDP by setting w_{acc} to 0.8 in the FF, leading to the model shown in Fig. 6 (b). This EDP-focused model achieves a $3.6 \times -21.2 \times$ reduction in EDP compared to previous methods, with only five hours of search time, and provides a 4.69% accuracy improvement over the EDP-optimized model from Gibbon. Figure 5 shows the accuracy-EDP trade-off obtained for various w_{acc} values in our FF. Overall, CrossNAS establishes a new benchmark for multi-objective optimization of PIM-based architectures, surpassing previous methods.

We also compare the models optimized by CrossNAS with other well-known deep learning models, such as AlexNet [19], VGG16

Table 4: Performance comparison against various deep learning models on CIFAR-100 Dataset.

Method	PIM Accuracy	Energy (mJ)	Latency (ms)	EDP ($mJ \times ms$)	Area (mm^2)
AlexNet [19]	57.1%	0.38	1.00	0.38	103.99
VGG16 [25]	67.2%	2.68	6.44	17.25	499.57
ResNet18 [12]	72.3%	1.33	3.59	4.76	466.94
CrossNAS ($w_{acc}=0.99$)	69.55%	0.81	3.62	2.93	282.12
CrossNAS ($w_{acc}=0.8$)	60.09%	0.193	1.096	0.211	343.436

[25] and ResNet18 [12], as presented in Table 3. The accuracy-optimized model achieves an accuracy improvement of 1.57%-9.57% compared to these models, along with an EDP reduction of $1.35 \times -61.5 \times$. The EDP-focused model delivers an EDP reduction of $5.2 \times -235.9 \times$, with a slight accuracy loss compared to VGG16 and ResNet18.

4.4 Performance Comparison Results on CIFAR-100 Dataset

Here, we analyze the performance of CrossNAS in optimizing models for the CIFAR-100 dataset and compare the results with those of well-known deep learning models, as listed in Table 4. Similar to CIFAR-10, we set the w_{acc} to 0.99 to obtain an accuracy-focused model, leading to the model shown in Fig. 6 (c). The accuracy-optimized model achieves a 12.45% accuracy improvement over AlexNet, and a 2.35% improvement over VGG16, but experiences a 2.75% accuracy drop compared to ResNet18. The accuracy drop occurs because ResNet18 has channel numbers as high as 512, whereas our designed search space limits the maximum channel number to 128 to optimize EDP. Therefore, the accuracy-focused model achieves a $1.62 \times$ reduction in EDP compared to ResNet18. Additionally, by setting w_{acc} to 0.8, we obtain an EDP-focused model, shown in Fig. 6 (d), which results in a $1.8 \times -22.6 \times$ reduction in EDP compared to other models, along with a 3% accuracy improvement over AlexNet. Note that a stride of 2 is used in the first 3×3 and 1×1 conv layers of the RES block when optimizing the models for CIFAR-100 by CrossNAS.

4.5 Analysis of Selected Models

By analyzing various models optimized with different w_{acc} values in the fitness function, we gain important insights that aid in designing PIM-specific model architectures tailored to particular design objectives.

Based on our observations from the optimized models selected by CrossNAS, shown in Fig. 6, the EDP-focused models for the CIFAR-10 dataset typically select the VGG block as the first block, as it reduces the feature map size through max pooling. This reduction in feature map size lowers the hardware cost in subsequent layers, thereby reducing the overall EDP. For the CIFAR-100 dataset, the EDP-focused models tend to choose either the RES ($stride = 2$) or VGG block as the first block, as both reduce the feature map size through higher stride or pooling layers.

From the mixed-precision quantization map, highlighted by various colors in Fig. 6, we observe that the EDP-focused models selected by CrossNAS tend to use higher weight bit widths in the middle convolutional layers, while the head and tail layers have lower weight bit precision. The activation bit width is generally

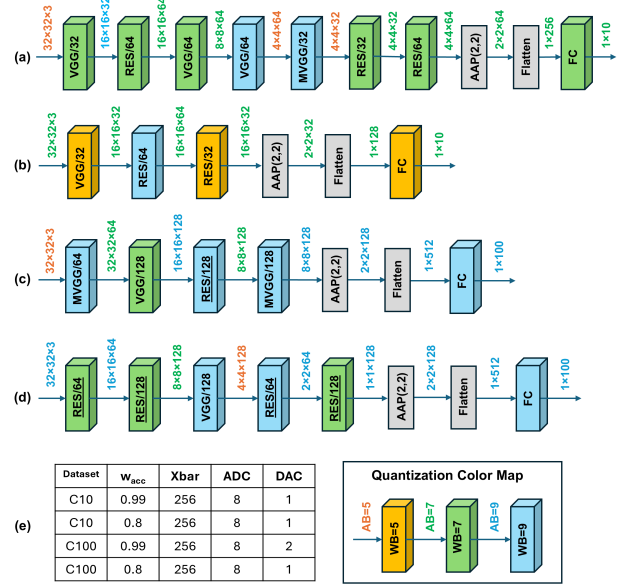


Figure 6: Block types along with channel numbers ($Block/K$) selected by CrossNAS for (a) CIFAR-10 dataset with $w_{acc} = 0.99$ and (b) $w_{acc} = 0.8$; (c) CIFAR-100 dataset with $w_{acc} = 0.99$ and (d) $w_{acc} = 0.8$; (e) corresponding PIM circuit configuration and quantization color map. AAP denotes adaptive average pooling, WB and AB denote weight and activation bit width, respectively. Underlined blocks indicate $stride = 2$.

lower in the middle layers, whereas the head and tail layers exhibit higher activation precision. For shallower models, e.g., Fig. 6 (b), there is a tendency to apply the same activation bit width across the entire architecture.

From the PIM circuit configuration selected by CrossNAS, shown in Fig. 6 (e), we observe that the EDP-focused models prefer a crossbar size of 256×256 . This configuration, combined with an 8-bit ADC, offers the best balance between accuracy and EDP. Most of the EDP-focused models opt for a 2-bit DAC, resulting in minimal accuracy loss, while improving both latency and energy efficiency due to the reduced number of digital-to-analog conversions.

5 Conclusion

In this paper, we present CrossNAS, an efficient weight-sharing-based NAS framework for PIM systems. We construct a multi-dimensional cross-layer search space that includes diverse architectures, layer-specific weight and activation bit widths, and PIM parameters such as crossbar size and ADC/DAC precisions. Our search leverages a multi-objective fitness function to explore various neural network architectures and PIM hardware configurations. CrossNAS outperforms previous PIM-oriented NAS methods and most of the well-known deep learning models, identifying superior models in terms of both accuracy and EDP.

Acknowledgments

This work is supported in part by the National Science Foundation (NSF) under grant number 2409697.

References

- [1] Md Hasibul Amin, Mohammed Elbity, and Ramtin Zand. 2022. Interconnect Parasitics and Partitioning in Fully-Analog In-Memory Computing Architectures. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 389–393. doi:10.1109/ISCAS48785.2022.9937884
- [2] Md Hasibul Amin, Mohammed E. Elbity, and Ramtin Zand. 2022. Xbar-Partitioning: A Practical Way for Parasitics and Noise Tolerance in Analog IMC Circuits. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12, 4 (2022), 867–877. doi:10.1109/JETCAS.2022.3222966
- [3] Md Hasibul Amin, Mohammadreza Mohammadi, and Ramtin Zand. 2024. Multi-Objective Neural Architecture Search for In-Memory Computing. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 343–348. doi:10.1109/ISVLSI61997.2024.00069
- [4] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R. Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, and Dejan S. Milojevic. 2019. PUMA: A Programmable Ultra-Efficient Memristor-Based Accelerator for Machine Learning Inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (Providence, RI, USA) (ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 715–731. doi:10.1145/3297858.3304049
- [5] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and Simplifying One-Shot Architecture Search. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*. Jennifer Dy and Andreas Krause (Eds.). PMLR, 550–559. <https://proceedings.mlr.press/v80/bender18a.html>
- [6] Hadjer Benmeziene, Corey Lammie, Irem Boybat, Malte Rasch, Manuel Le Gallo, Hsinyu Tsai, Ramachandran Muralidhar, Smail Niar, Ouarnoughi Hamza, Vijay Narayanan, Abu Sebastian, and Kaoutar El Maghraoui. 2023. AnalogNAS: A Neural Network Design Framework for Accurate Inference with Analog In-Memory Computing. In *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. 233–244. doi:10.1109/EDGE60047.2023.00045
- [7] Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. arXiv:1812.00332 [cs.LG] <https://arxiv.org/abs/1812.00332>
- [8] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory. In *Proceedings of the 43rd International Symposium on Computer Architecture (Seoul, Republic of Korea) (ISCA '16)*. 27–39.
- [9] Mohammed Elbity, Abhishek Singh, Brendan Reidy, Xiaochen Guo, and Ramtin Zand. 2021. An In-Memory Analog Computing Co-Processor for Energy-Efficient CNN Inference on Mobile Devices. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. 188–193. doi:10.1109/ISVLSI51109.2021.00043
- [10] Mohammed E. Elbity, Brendan Reidy, Md Hasibul Amin, and Ramtin Zand. 2023. Heterogeneous Integration of In-Memory Analog Computing Architectures with Tensor Processing Units. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (Knoxville, TN, USA) (GLSVLSI '23)*. Association for Computing Machinery, New York, NY, USA, 607–612. doi:10.1145/3583781.3590256
- [11] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single Path One-Shot Neural Architecture Search with Uniform Sampling. In *Computer Vision – ECCV 2020*, Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (Eds.). Springer International Publishing, Cham, 544–560.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778. doi:10.1109/CVPR.2016.90
- [13] Miao Hu, John Paul Strachan, Zhiyong Li, Emmanuelle M. Grafals, Noraica Davila, Catherine Graves, Sity Lam, Ning Ge, Jianhua Joshua Yang, and R. Stanley Williams. 2016. Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication. In *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. doi:10.1145/2897937.2898010
- [14] Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. 2021. Device-Circuit-Architecture Co-Exploration for Computing-in-Memory Neural Accelerators. *IEEE Trans. Comput.* 70, 4 (2021), 595–605. doi:10.1109/TC.2020.2991575
- [15] Donghyuk Kim, Chengshuo Yu, Shanshan Xie, Yuzong Chen, Joo-Young Kim, Bongjin Kim, Jaydeep P. Kulkarni, and Tony Tae-Hyoung Kim. 2022. An Overview of Processing-in-Memory Circuits for Artificial Intelligence and Machine Learning. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 12, 2 (2022), 338–353. doi:10.1109/JETCAS.2022.3160455
- [16] Sungho Kim, Hee-Dong Kim, and Sung-Jin Choi. 2019. Impact of Synaptic Device Variations on Classification Accuracy in a Binarized neural network. *Scientific reports* 9, 1 (2019), 1–7.
- [17] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] <https://arxiv.org/abs/1412.6980>
- [18] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report 0. University of Toronto, Toronto, Ontario. <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf
- [20] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Gieffers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. 2018. Mixed-precision in-memory computing. *Nature Electronics* 1, 4 (2018), 246–253.
- [21] Shubham Negi, Indranil Chakraborty, Aayush Ankit, and Kaushik Roy. 2022. NAX: neural architecture and memristive xbar based accelerator co-design. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (San Francisco, California) (DAC '22)*. Association for Computing Machinery, New York, NY, USA, 451–456. doi:10.1145/3489517.3530476
- [22] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameters Sharing. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*. Jennifer Dy and Andreas Krause (Eds.). PMLR, 4095–4104. <https://proceedings.mlr.press/v80/pham18a.html>
- [23] Sebastian Ruder. 2017. An overview of gradient descent optimization algorithms. arXiv:1609.04747 [cs.LG] <https://arxiv.org/abs/1609.04747>
- [24] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramanian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. 2016. ISAAC: A Convolutional Neural Network Accelerator with in-Situ Analog Arithmetic in Crossbars (ISCA '16). IEEE Press, 14–26. doi:10.1109/ISCA.2016.12
- [25] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations*.
- [26] Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. 2020. Single-Path NAS: Designing Hardware-Efficient ConvNets in Less Than 4 Hours. In *Machine Learning and Knowledge Discovery in Databases*, Ulf Brefeld, Elisa Fromont, Andreas Hotho, Arno Knobbe, Marloes Maathuis, and Céline Robardet (Eds.). Springer International Publishing, Cham, 481–497.
- [27] Hanbo Sun, Zhenhua Zhu, Yi Cai, Xiaoming Chen, Yu Wang, and Huazhong Yang. 2020. An Energy-Efficient Quantized and Regularized Training Framework For Processing-In-Memory Accelerators. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 325–330. doi:10.1109/ASP-DAC47756.2020.9045192
- [28] Hanbo Sun, Zhenhua Zhu, Chenyu Wang, Xuefei Ning, Guohao Dai, Huazhong Yang, and Yu Wang. 2023. Gibbon: An Efficient Co-Exploration Framework of NN Model and Processing-In-Memory Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 11 (2023), 4075–4089. doi:10.1109/TCAD.2023.3262201
- [29] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yangan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 10726–10734. doi:10.1109/CVPR.2019.01099
- [30] Zheyu Yan, Da-Cheng Juan, Xiaobo Sharon Hu, and Yiyu Shi. 2021. Uncertainty Modeling of Emerging Device based Computing-in-Memory Neural Accelerators with Application to Neural Architecture Search. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 859–864.
- [31] Zhihang Yuan, Jingze Liu, Xingchen Li, Longhao Yan, Haoxiang Chen, Bingzhe Wu, Yuchao Yang, and Guangyu Sun. 2021. NAS4RRAM: neural network architecture search for inference on RRAM-based accelerators. *Science China Information Sciences* 64, 6 (10 May 2021), 160407. doi:10.1007/s11432-020-3245-7
- [32] Ramtin Zand, Arman Roohi, and Ronald F DeMara. 2018. Fundamentals, modeling, and application of magnetic tunnel junctions. In *Nanoscale Devices*. CRC Press, 337–368.
- [33] Xinbang Zhang, Zehao Huang, Naiyan Wang, Shiming Xiang, and Chunhong Pan. 2021. You Only Search Once: Single Shot Neural Architecture Search via Direct Sparse Optimization. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 43, 09 (Sept. 2021), 2891–2904. doi:10.1109/TPAMI.2020.3020300
- [34] Zhenhua Zhu, Hanbo Sun, Tongxin Xie, Yu Zhu, Guohao Dai, Lixue Xia, Dimin Niu, Xiaoming Chen, Xiaobo Sharon Hu, Yu Cao, Yuan Xie, Huazhong Yang, and Yu Wang. 2023. MNSIM 2.0: A Behavior-Level Modeling Tool for Processing-In-Memory Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 11 (2023), 4112–4125. doi:10.1109/TCAD.2023.3251696