AutoHet: An Automated Heterogeneous ReRAM-Based Accelerator for DNN Inference

Tong Wu[*]
Zhejiang University
Hangzhou, China
wu.tong@zju.edu.cn

Shuibing He[†]
Zhejiang University
Hangzhou, China
heshuibing@zju.edu.cn

Jianxin Zhu
Zhejiang University
Hangzhou, China
zjxin@zju.edu.cn

Weijian Chen
Zhejiang University
Hangzhou, China
weijianchen@zju.edu.cn

Siling Yang
Zhejiang University
Hangzhou, China
slingzjunet@zju.edu.cn

Ping Chen
Zhejiang University
Hangzhou, China
zjuchenping@zju.edu.cn

Yanlong Yin
Zhejiang University
Hangzhou, China
yinyanlong@gmail.com

Xuechen Zhang
Washington State University Vancouver
Vancouver, WA, USA
xuechen.zhang@wsu.edu

Xian-He Sun
Illinois Institute of Technology
Chicago, IL, USA
sun@iit.edu

Gang Chen
Zhejiang University
Hangzhou, China
cg@zju.edu.cn

ABSTRACT

ReRAM-based accelerators have become prevalent in accelerating deep neural network inference owing to their in-situ computing capability of ReRAM crossbars. However, most existing ReRAM-based accelerators are designed with homogeneous crossbars, leading to either low resource utilization or sub-optimal energy efficiency. In this paper, we propose AutoHet, an automated heterogeneous ReRAM-based accelerator with varied-size crossbars for different DNN layers. To achieve both high crossbar utilization and energy efficiency, AutoHet uses a reinforcement learning algorithm to automatically determine the proper crossbar configuration for each DNN layer. Additionally, AutoHet introduces rectangle crossbars and a tile-shared crossbar allocation scheme to reduce crossbar wastage and energy consumption. Experiment results show that AutoHet effectively improves crossbar utilization by up to 3.1× and reduces energy consumption by up to 94.6%, compared to approaches with homogeneous ReRAM crossbars.

CCS CONCEPTS

• **Computer systems organization → Heterogeneous (hybrid) systems**; • **Hardware → Emerging architectures**.

[*]Also with Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China.
[†]Shuibing He is the corresponding author.

KEYWORDS

Processing-in-memory, ReRAM-based accelerator, Heterogeneous architecture, Reinforcement learning

ACM Reference Format:
Tong Wu, Shuibing He, Jianxin Zhu, Weijian Chen, Siling Yang, Ping Chen, Yanlong Yin, Xuechen Zhang, Xian-He Sun, and Gang Chen. 2024. AutoHet: An Automated Heterogeneous ReRAM-Based Accelerator for DNN Inference. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3673038.3673143

1 INTRODUCTION

Deep neural networks (DNNs) have been widely used in various fields of modern society [22]. Typically DNNs include convolutional layers, which involve massive matrix-vector multiplications (MVMs). Since MVMs cause a large number of data movement between processor and memory in traditional von Neumann architectures [22, 30], DNN inference exhibits high computational latency and energy consumption. These two issues are exacerbated facing the ever-increasing network sizes and user performance demands [5]. Recently, an emerging processing-in-memory (PIM) technology, resistive random access memory (ReRAM), has been proposed to reduce latency and energy overhead by performing MVMs in an in-situ computing manner [16]. The ReRAM-based accelerators gradually become prevalent for DNN inference [2, 19, 21, 27].

The fundamental unit of ReRAM-based accelerators is the crossbar. An $m \times n$ crossbar consists of m wordlines on rows, n bitlines on columns, and $m \times n$ memristors as cells [16]. Prior to DNN inference, the model's weights are pre-loaded into the crossbar cells as conductance values. During the inference process, input signals are transformed into voltage levels on the wordlines. These voltages are subsequently multiplied by the conductances of the cells, generating currents on the bitlines in accordance with Ohm's Law. The currents on the same bitline are summed and converted to a digital signal, completing a multiply-accumulate (MAC) operation [16].

Multiple bitlines on a crossbar are computed to perform the MVMs. Besides crossbars, accelerators also require peripheral circuits (e.g., DACs, ADCs) to perform the whole inference process [16].

Crossbar utilization and energy consumption are two important system metrics for evaluating the efficiency of ReRAM-based accelerators [2, 19, 27], especially in mobile environments where hardware resource and system energy are extremely limited [7, 24]. Low crossbar utilization indicates a wastage of storage and computing resources, which affects operating speed and even causes task execution failure. High energy consumption increases computing costs and hinders the advantage of ReRAM. Therefore, to better leverage ReRAM-based accelerators, we pursue both high crossbar utilization and energy efficiency. However, most of the existing ReRAM-based accelerators [2, 19, 21] struggle to simultaneously achieve the above goals for two reasons.

First, the homogeneous crossbar architecture may render sub-optimal crossbar utilization or energy efficiency. Existing ReRAM-based accelerators usually adopt a crossbar-level homogeneous architecture, which uses fixed-size crossbars for the entire model [2, 19, 21]. Due to the diverse computational characteristics of individual DNN layers, a specific crossbar size may work well for one layer but may not be universally applicable for all the layers. Therefore, homogeneous crossbars can hardly simultaneously achieve high utilization and energy goals for the whole DNN model.

Second, the tile-based crossbar allocation scheme may compromise resource utilization. The existing ReRAM-based accelerators adopt a hierarchical topology that integrates a fixed number of crossbars into one tile and uses the tile as the minimum allocation unit for each DNN layer [19, 31]. This *tile-based* policy involves allocating the entire tile to a layer, even if the layer is too small to occupy all the crossbars within the tile, potentially resulting in crossbar wastage. For large layers, the policy employs a round-up approach to allocate tiles. Both cases lead to sub-optimal crossbar utilization.

To overcome these issues, we propose a crossbar-level heterogeneous ReRAM architecture for DNN inference. Unlike existing architectures, it uses varied-size crossbars for different layers of the DNN model. However, providing this feature is quite challenging for two reasons. First, for each DNN layer, selecting a suitable crossbar size to achieve both high utilization and energy efficiency is difficult. This is because the two objectives are usually in conflict: small crossbars bring high crossbar utilization but they invoke more PCs (e.g., ADCs), thus increasing energy consumption; vice versa. Second, the decision space for determining crossbar sizes for all DNN layers may be vast, considering the numerous layers in a DNN model and the variety of candidate crossbar sizes. Therefore, we need an automated approach to perform fast and efficient searches, as a manual approach is time-consuming, expertise-requiring, and error-prone.

To address the above challenges, we design AutoHet, an automated heterogeneous ReRAM-based accelerator for DNN inference. AutoHet utilizes a reinforcement learning (RL) algorithm to automatically determine the heterogeneous crossbar size for each DNN layer. The RL algorithm expedites the optimal strategy search by learning the features of different DNN layers. It also achieves both high crossbar utilization and energy efficiency by integrating these factors into the reward function. Additionally, it introduces hybrid crossbar shapes (i.e., square and rectangle crossbars) as candidates

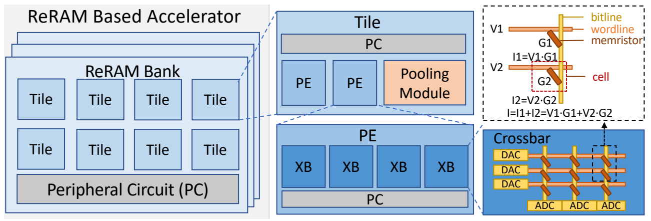


Figure 1: The generic architecture of ReRAM-based accelerators.

for RL search to further improve crossbar utilization. Furthermore, AutoHet proposes a tile-shared crossbar allocation scheme that allocates multiple layers to one tile to reduce crossbar wastage.

In summary, this paper makes the following contributions:

- We propose a crossbar-level heterogeneous ReRAM-based accelerator to achieve high crossbar utilization and energy efficiency simultaneously for DNN inference.
- We introduce an RL algorithm to automatically determine the heterogeneous crossbar sizes for individual DNN layers.
- We propose the tile-shared crossbar allocation scheme that allocates multiple layers to one tile to reduce crossbar wastage.
- We evaluate AutoHet with extensive experiments. Our results show that AutoHet improves crossbar utilization by up to 3.1× and reduces energy consumption by up to 94.6%, compared to state-of-the-art approaches.

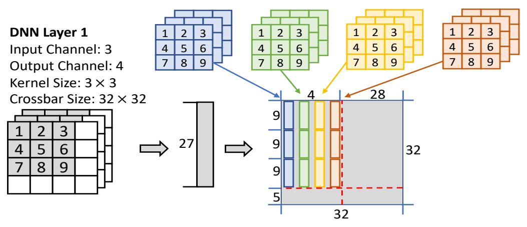
2 BACKGROUND AND MOTIVATION

2.1 ReRAM-based Accelerator Architecture

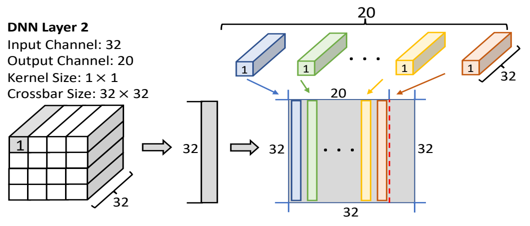
Architecture overview. ReRAM is an emerging device with high-density, low-latency, low-energy storage, and in-memory computing capabilities [16]. Figure 1 illustrates the generic architecture of ReRAM-based accelerators. It comprises multiple ReRAM banks, each containing numerous tiles. Each tile consists of several processing elements (PEs), and each PE integrates multiple crossbars. Crossbars and peripheral circuits (e.g., DACs, ADCs, shift and adders, pooling modules) cooperate to perform DNN inference together.

In-situ computing. The crossbar serves as the computing and storage unit to perform highly-parallel in-situ computing. A crossbar consists of wordlines on rows, bitlines on columns, and memristors that store the DNN weights as conductance G values. Each bitline is orthogonally connected to each wordline through a memristor and forms a basic computing unit, a cell. Input data is converted to the voltage V values on wordlines through DACs and each bitline produces the current I according to Ohm's law (i.e., $I = V \times G$). The sum of the multiple currents on a bitline represents the result of a MAC operation. The in-situ MVM computing is completed after converting all the currents of all bitlines into digital signals through ADCs.

DNN layer mapping and crossbar allocation. Owing to the low computational latency and energy consumption, ReRAM-based accelerators are widely employed in DNN inferences. A DNN model consists of multiple layers, where each layer contains multiple kernels. Typically, these kernels are mapped onto the crossbars in sequence. Figure 2 shows an example of how to map kernels of two different DNN layers onto a 32×32 crossbar. For simplicity, we do



(a) Four $3 \times 3 \times 3$ kernel matrices in layer 1



(b) Twenty $1 \times 1 \times 32$ kernel matrices in layer 2

Figure 2: The mapping between DNN layers and 32×32 crossbars.

not consider weight quantization. The kernel sizes of layer 1 and layer 2 are 3×3 and 1×1, respectively. Since the number of input and output channels of layer 1 is three and four, this layer needs to map four $3 \times 3 \times 3$ kernel matrices onto the crossbar. Each kernel matrix is first expanded into a column vector and then mapped onto a column of the crossbar. Therefore, layer 1 requires a total of four columns of the crossbars to store its kernels. Likewise, layer 2 requires 20 columns to store its kernel parameters. If one crossbar is insufficient to accommodate all the kernels of the layer, another crossbar in the tile will be occupied. Note that a tile only stores the weights of a single DNN layer in existing crossbar allocation and mapping schemes [19, 31]. When all kernel mappings are completed, the tile is ready to perform MVM calculations.

2.2 Motivation and Challenges

ReRAM-based accelerators are commonly deployed in mobile and edge environments, where the chip area and battery power are very limited [7, 24]. Therefore, it is very important to simultaneously achieve high crossbar utilization and energy efficiency. However, existing ReRAM-based accelerators are difficult to simultaneously meet both of these criteria, as they have the following two limitations.

2.2.1 The Homogeneous Crossbar Architecture Causes Sub-Optimal Resource Utilization or Energy Efficiency.

Existing ReRAM-based accelerators adopt a homogeneous architecture, namely, they consist of a single size of crossbars [2, 19, 21]. Since each DNN layer has different computing characteristics, one type of crossbars may be suitable for one layer of the DNN models, but not for all of them. We still use Figure 2 as an example to further verify this. It shows the results of mapping two DNN layers onto the same 32×32 crossbar. The crossbar utilization is 10.5% for layer 1 and 62.5% for layer 2. This implies that, for crossbar utilization, the 32×32 crossbar is suitable for layer 2 but not for layer 1. Similarly, the suitability of crossbars for energy efficiency varies for different DNN layers. Therefore, mapping the entire DNN model

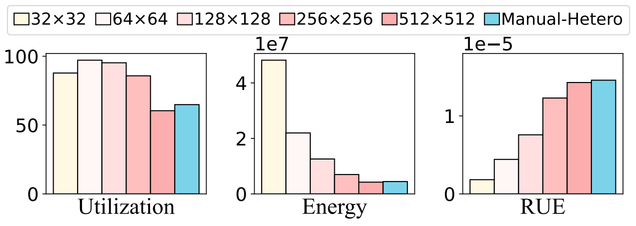


Figure 3: The performance of homogeneous and heterogeneous crossbars.

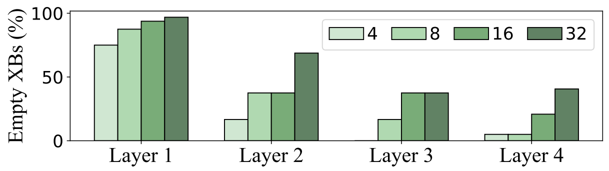


Figure 4: The proportion of empty crossbars. The legends denote the number of crossbars contained in one tile.

onto the homogeneous crossbars fails to achieve the best result for the whole DNN model.

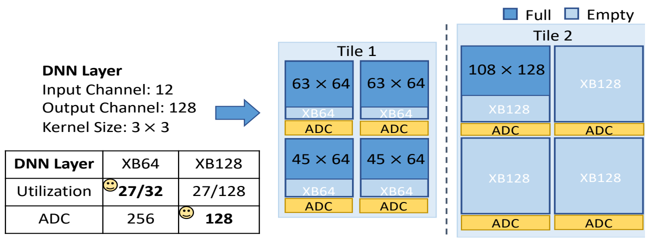
Since most of the existing metrics are one-dimensional criteria and thus make it hard to quantify the system performance, we propose a new metric, Ratio of Utilization and Energy (RUE), to consider the two metrics jointly. RUE is defined as U/E, where U and E denote the crossbar utilization and the energy consumption, respectively. Figure 3 shows the results of VGG16 when mapped onto accelerators with homogeneous and heterogeneous crossbars. For the homogeneous accelerators, we choose five crossbar sizes, 32×32, 64×64, 128×128, 256×256, and 512×512, as commonly used in related work[19, 27, 29]. For the heterogeneous accelerator, we manually set the crossbar size as 512×512 for the first ten layers of VGG16 and 256×256 for the last six layers. It shows that the homogeneous accelerators can only achieve either high resource utilization (i.e., with 32×32) or low energy consumption (i.e., with 512×512), leading to low RUE values. In contrast, the manual heterogeneous accelerator obtains the highest RUE values.

Furthermore, we also observe that the existing crossbars are usually square, and their side lengths are powers of 2, which are not well-matched with the odd side lengths of some kernels (e.g., 1×1 and 3×3). As a result, significant internal wastage of the crossbar occurs when mapping these kernels onto the square crossbars, as shown in Figure 2. This will also lower the RUE values. Rectangle crossbars help to address this issue (see §3.3).

The above two points motivate us to propose heterogeneous accelerators with varied-size/shape crossbars for DNN layers to optimize both the crossbar utilization and energy efficiency for the whole model.

2.2.2 The Tile-Based Allocation Scheme Compromises Resource Utilization.

Most of the current ReRAM-based accelerators utilize the tile as the basic crossbar allocation unit for each DNN layer and allocate one or more tiles to them [19, 31]. Each tile can only accommodate kernels from one DNN layer to simplify the data flow. However, this *tile-based* scheme can lead to crossbar wastage, thus lowering resource utilization. For example, let us assume a tile has four 64×64 crossbars. Then, for a small DNN layer (as shown in Figure 2), it



DNN Layer	XB64	XB128
Utilization	**27/32**	27/128
ADC	256	**128**

Figure 5: The comparison of the same DNN layer mapping onto 64×64 and 128×128 crossbars.

only needs one crossbar to accommodate its kernel data. With the tile-based scheme, one tile will be allocated for the layer and three crossbars will be idle. Therefore, the crossbar wastage is 75%. This problem still exists for large layers. For instance, assuming a large layer requires five crossbars, it will be assigned two tiles, resulting in an overall crossbar wastage of 3/8=37.5%.

Figure 4 illustrates the empty crossbar proportion of four DNN layers of VGG16 on a ReRAM-based accelerator with four tile configurations. The crossbar size is fixed to 64×64, and the number of crossbars per tile varies from 4 to 32. It reveals that only 58% of the crossbars are utilized on average across these layers. Furthermore, the proportion of empty crossbars increases with the increase of the tile size. Specifically, the average empty crossbar ratio across these layers is 24% when a tile contains four crossbars, while the ratio rises to 60% when a tile contains 32 crossbars. In a nutshell, the existing tile-based scheme tends to reduce the utilization of crossbars, thereby decreasing the RUE value.

2.2.3 *Challenges.*

To overcome the limitations of existing ReRAM-based accelerators, we need to use crossbars with different sizes for each DNN layer. This introduces two challenges.

First, for a specific DNN layer, selecting a suitable crossbar size to achieve both high utilization and low energy consumption is difficult. This is because these two objectives are usually conflict: small crossbars result in high crossbar utilization but require more PCs (e.g., ADC), leading to increased energy consumption, and vice versa [29]. Figure 5 shows the utilization and the number of activated ADCs when mapping 128 $3 \times 3 \times 12$ kernel matrices onto a 64×64 crossbar and a 128×128 crossbar. As expected, the former crossbar achieves higher utilization, whereas the latter consumes less energy due to fewer activated ADCs.

Second, the search space for determining crossbar sizes for all DNN layers may be vast, considering the numerous layers in a DNN model and the variety of crossbar sizes that can be chosen from. For a DNN model with N layers, assuming there are C crossbar candidates for each layer, the size of the solution space is C^N. A manual approach is time-consuming, expertise-requiring, and error-prone. Therefore, we need an automated approach that can effectively explore the search space and find the proper crossbar size for each DNN layer.

3 DESIGN OF AUTOHET

3.1 System Overview

We propose AutoHet, an automated crossbar-level heterogeneous ReRAM-based accelerator for optimizing DNN inferences. Figure 6 shows the overview framework of AutoHet, which consists of the RL model and the heterogeneous ReRAM-based accelerator. The RL model produces the heterogeneous crossbar configurations for DNN models. The heterogeneous ReRAM-based accelerator is built based on the crossbar configuration and performs DNN inference. To choose the best configuration for the DNN model, the accelerator outputs direct hardware feedback as a reward to optimize the RL model.

The RL model. Reinforcement learning [17] is a well-established machine learning method for automatically dealing with the neural architecture search problems [23]. Therefore, we choose the RL model as the core scheme to search the heterogeneous crossbar configurations for the whole DNN model. We construct the RL agent based on the deep deterministic policy gradient (DDPG) [20] algorithm, which includes paired actor and critic networks, for solving RL problems in continuous state and action space. The actor predicts an action according to the given state (e.g., kernel size, input channels), while the critic evaluates the importance of the action-state pair using the Q-function [20]. The state space records the features of DNN models and provides an observation for the RL agent. The reward function evaluates the performance of the crossbar strategy based on the hardware feedbacks. The experience pool collects the states, actions, and rewards to optimize the pair-network regularly.

Heterogeneous ReRAM-based accelerator. For simplicity, the accelerator is built still with the traditional hierarchical architecture (i.e., ReRAM bank, tile, PE, crossbar), as shown on the right side of Figure 6. However, it has three differences from the traditional architecture. First, while crossbars in a tile are homogeneous, crossbars in different tiles can be heterogeneous. Second, to further improve the internal utilization within a crossbar, the crossbar does not necessarily to be square; it can be rectangular. Third, to address the crossbar wastage within a tile, the accelerator allows mapping multiple DNN layers onto the same tile to reuse the empty crossbars in allocated tiles. Notably, both the RL-based determination process and the tile-shared scheme run on the CPU side. We use a global controller (GC) to decode CPU instructions and control the heterogeneous DNN mapping and inference. The GC receives instructions and signals the input/output buffer and tiles through the bus[2, 29].

3.2 RL-based Heterogeneous Crossbar Configuration Decision Scheme

We model the optimized layer-wise heterogeneous crossbar configuration strategy as the RL learning target. The action of the RL agent corresponds to the specific crossbar type (size) for each DNN layer. We adjust the reward function to balance the crossbar utilization and energy consumption. The RL agent will automatically search the proper crossbar type for each DNN layer.

Work flow. The working process of the RL model is numbered from ① to ⑫ in Figure 6. The solid arrows denote the decision stage of generating the crossbar configuration, while the dashed arrows indicate the internal learning stage of the RL model. In the beginning, the state space updates based on the features of each DNN layer (①). Then, the RL agent produces an action (i.e., the crossbar type of the current layer) according to the state space (②

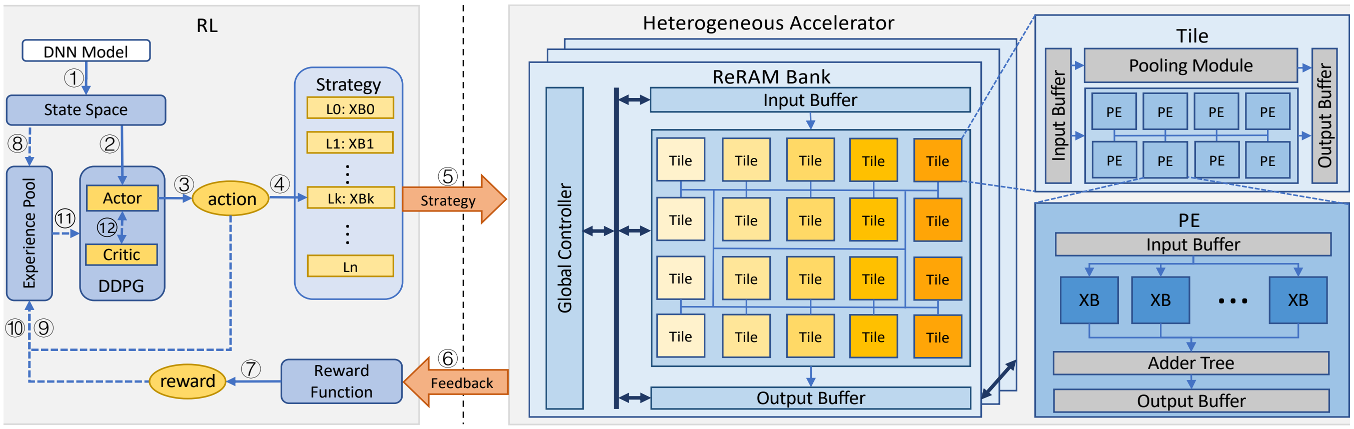

RL
DNN Model
① State Space
② ⑧ Experience Pool ⑪ Actor ⑫ Critic DDPG
③ action ④
Strategy L0: XB0 L1: XB1 Lk: XBk Ln
⑤ Strategy
⑩ ⑨ reward ⑦ Reward Function ⑥ Feedback
Heterogeneous Accelerator
ReRAM Bank
Global Controller
Input Buffer
Tile
Output Buffer
Tile
Input Buffer Pooling Module Output Buffer PE
PE
Input Buffer
XB XB XB
Adder Tree
Output Buffer

Figure 6: The overview framework of AutoHet.

~ ③). These three processes will be repeated until all DNN layers receive actions. All the actions form a heterogeneous crossbar configuration in order (④). The heterogeneous ReRAM-based accelerator performs DNN mapping and inference depending on the configuration (⑤). The feedbacks (i.e., crossbar utilization and energy consumption) are sent to the reward function to generate a reward value (⑥ ~ ⑦). So far, the decision stage of the RL model is suspended and the learning stage begins. The experience pool collects the states, actions, and rewards generated in the previous steps (⑧ ~ ⑩). Then the RL agent samples a batch of experiences and updates the pair-network in the background (⑪ ~ ⑫). The decision stage and learning stage are alternately executed in an offline manner until the optimal crossbar configuration strategy is found. Then we choose the optimal strategy as the final solution to perform DNN mapping and inference.

Action space. To realize the layer-wise DNN inference on the crossbar-level heterogeneous architecture, we choose crossbar type as the action. The action a_k denotes the crossbar type of the layer k, and the accelerator will map the k-th DNN layer onto tiles with this type of crossbar. We prepare different types of crossbars as the candidates and map them to a sequence of integers to formulate the action space. The details of selecting the crossbar candidates are discussed in §3.3.

State space. To fully utilize the characteristics of each DNN layer, we define a 10-dimensional state vector S as our observation. The state vector of the layer k can be exhibited as follows:

$$S_k = (k, t, inc, outc, ks, s, w, ins, a_k, u_k) \tag{1}$$

All the states are defined in Table 1. Notably, we consider the FC layer as a special kind of CONV layer by setting both ks and s to one and defining their inc and $outc$ as the number of input and output neurons, respectively [27, 28]. The state vector consists of eight static features directly obtained from each DNN layer and two dynamic features (i.e., a_k and u_k) obtained from the decision stage of the RL agent. The a_k denotes the action of the current step (i.e., the crossbar type of layer k). The u_k is the crossbar utilization of layer k, which can be calculated through a_k and static features, as Equation 4 shows.

Reward function. The reward R is used to evaluate the effectiveness of the RL agent. It needs to simultaneously consider the crossbar utilization and energy consumption. We use the direct hardware feedbacks of the heterogeneous ReRAM-based accelerator to compose the reward function as follows:

Table 1: Symbols used in the RL state space.

No.	Symbol	Meaning
1	k	layer index
2	t	layer type: CONV:1 ; FC: 0
3	inc	number of channels in the input feature map
4	$outc$	number of channels produced by the CONV
5	ks	number of elements of a convolution kernel
6	s	stride of the convolution
7	w	number of weights in layer k
8	ins	size of the input feature map
9	a_k	action of layer k
10	u_k	crossbar utilization of layer k

$$R = \frac{u}{e} \tag{2}$$

where u and e are the crossbar utilization and energy consumption, respectively. Because the order of magnitude of energy consumption is much greater than that of the crossbar utilization, the value of reward R is between $[0, 1]$, which is conducive to the convergence of the RL model.

Experience pool. Once the DNN inference is finished, the experience pool collects actions, states, and the reward of this process. The RL agent samples from the experience pool to periodically update the pair-network and improve search efficiency. The experience of layer k can be abstracted as:

$$E_k = (S_k, S_{k+1}, a_k, R) \tag{3}$$

where S_k and S_{k+1} are the state space of layer k and $k+1$, a_k is the action of layer k and R is the reward of the inference.

3.3 Heterogeneous Crossbar Size Selection

To realize the high resource utilization within a crossbar, we need to carefully select the crossbar sizes to form the heterogeneous accelerator architecture. To this end, we formulate a mathematical model to calculate the crossbar utilization after mapping a DNN layer's kernel data to the ReRAM-based accelerator. Then, we apply this formula to guide the crossbar type selection for DNN layers.

Crossbar utilization formula. Assume a convolutional (CONV) layer in a DNN model has the kernel size of $k \times k$, input channels of

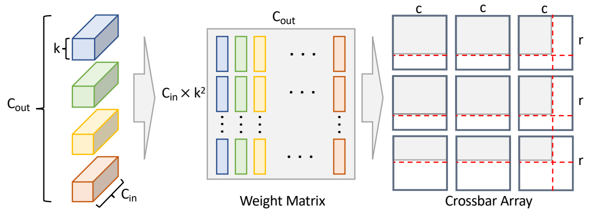


Figure 7: The mapping scheme between kernels and crossbars.

Table 2: The structure of three popular DNN models. The symbol $aCb - c$ represents that a CONV layers with $b \times b$-sized kernels and c output channels. The symbol Fd means a fully-connected layer with d neurons.

Network	Structure
AlexNet	C3-64, C3-192, C3-384, 2C3-256, F4096, F4096, F10
VGG16	2C3-64 , 2C3-128, 3C3-256, 6C3-512,F4096, F1000, F10
ResNet152	C7-64, 3C1-64, 8C1-128, 40C1-256, 12C1-512, 37C1-1024, 4C1-2048, 3C3-64, 8C3-128, 36C3-256, 3C3-512, F1000

C_{in}, and output channels of C_{out}. This layer will be mapped onto a crossbar array that contains multiple $r \times c$ crossbars, as shown in Figure 7. The layer is first unfolded into a weight matrix with $C_{in} \times k^2$ rows and C_{out} columns. Then the weight matrix will be mapped across multiple crossbars.

To ensure computational parallelism, we map the data from one single kernel onto a single crossbar. Because each $r \times c$ crossbar can store up to $\lfloor r/k^2 \rfloor$ kernels in row and c kernels in column, some cells within the crossbars cannot be fully utilized as shown in Figure 7. Furthermore, the whole crossbar array needs to include $\lceil C_{in}/\lfloor r/k^2 \rfloor \rceil$ rows of crossbars and $\lceil C_{out}/c \rceil$ columns of crossbars. Therefore, the overall utilization of the crossbar array can be calculated as Equation 4.

$$u = \frac{C_{in} \times k^2 \times C_{out}}{r \times \lceil C_{in}/\lfloor r/k^2 \rfloor \rceil \times c \times \lceil C_{out}/c \rceil} \tag{4}$$

For fully connected (FC) layers, this formula remains applicable by setting $k = 1$ and C_{in} and C_{out} as the number of input and output neurons of the FC layer.

Naive crossbar candidate selection. From Equation 4, it is evident that choosing the same crossbar size for each DNN layer is sub-optimal, as different DNN layers have varying values for C_{in}, C_{out}, and k. To enhance the utilization, a naive approach is to consider multiple commonly used crossbars as candidates, typically including 32×32, 64×64, 128×128, 256×256, and 512×512, as mentioned in previous work [19, 27, 29]. However, we notice that the existing crossbar candidate shapes are all squares with side lengths that are powers of 2, and they are not suitable for all DNN layers. Table 2 shows the weight sizes for each layer of three classical DNN models. We observe that square crossbars are suitable for FC layers with side lengths close to or equal to powers of 2 (e.g., FC 1000, FC 4096), minimizing wastage. In contrast, CONV layers with typical 3×3 kernels show low utilization on square crossbars with powers of 2 side lengths due to the mismatch. In summary, only using existing square heterogeneous crossbars still results in a large number of wasted cells within the crossbar.

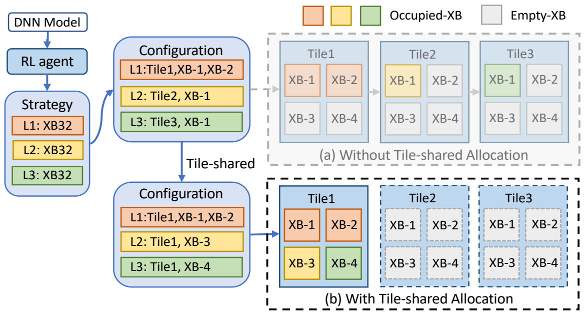


Figure 8: The tile-shared crossbar allocation scheme.

Hybrid crossbar shape selection. To make crossbar candidates suitable for both FC and CONV layers in a DNN model, it is necessary to choose both square and rectangle crossbars as candidates. To determine the desired sizes of rectangle crossbars, we analyze the weight matrix sizes by unfolding all CONV layers in common DNN models. We observe that most weight matrices misaligned with square crossbars come from the 3×3 kernel size. Specifically, it constitutes 62.5%, 81.25%, and 32.05% for AlexNet, VGG16, and ResNet152, respectively. Since the widths of these weight matrices are powers of 2, we only adjust the height of the rectangle crossbars to be multiples of 9, reducing the number of wasted rows within the crossbar. For example, the fourth layer of VGG16 (i.e., $k = 3, C_{in} = 128, C_{out} = 128$) achieves 83.7% utilization on 32×32 crossbars but achieves 100% on 36×32 crossbars. Following this principle, we design five heterogeneous crossbar candidates in our experiments (i.e., $32 \times 32, 36 \times 32, 72 \times 64, 288 \times 256, 576 \times 512$). Similarly, users can tailor heterogeneous crossbars based on the architecture of their target DNNs, which can also be supported by AutoHet (§4.4).

Overall, employing a mixture of crossbar shapes enhances the cell utilization of crossbars. Additionally, the increased crossbar utilization results in a reduced number of required crossbars and PCs, contributing to lower energy consumption and a high RUE value.

3.4 Tile-shared Crossbar Allocation Scheme

As discussed in §2.2, the existing tile-based allocation scheme of ReRAM-based accelerators may lead to significant crossbar wastage in a tile when the tile cannot be fully filled. To address this issue, we propose a tile-shared crossbar allocation scheme for ReRAM-based accelerators. The key idea is allowing multiple DNN layers to be mapped onto the same tile, thereby reducing the number of empty crossbars by tile sharing.

Figure 8 shows an example of the proposed tile-shared crossbar allocation scheme. The RL agent generates a crossbar mapping strategy that maps three DNN layers ($L1-L3$) onto 32×32 crossbars. We assume that each layer can fit into a single tile. Without the tile-shared scheme, the Global Controller directly maps $L1$, $L2$, and $L3$ onto Tile 1, Tile 2, and Tile 3, respectively, as shown in Figure 8(a). This mapping approach results in 8/12 of crossbars in the three tiles being wasted. In contrast, with the tile-shared scheme, as illustrated in Figure 8(b), the Global Controller first remaps the $L2$ and $L3$ layers onto the unoccupied crossbars in Tile 1, and then releases Tile 2 and Tile 3. Therefore, the utilization of Tile 1 is improved from 50%

to 100%, while Tiles 2 and 3 become available for other layers in the DNN model or other models.

Note that the tile selection for data sharing cannot be arbitrary. AutoHet assigns different DNN layers with crossbars of various sizes to maximize RUE values. Therefore, it is crucial to ensure that the selected tiles for sharing should have the same crossbar size. To this end, AutoHet first groups all the used tiles based on their crossbar sizes. The tiles within each group have the same crossbar size. Then, for each tile group, AutoHet executes Algorithm 1 to obtain the tile combinations where data needs to be remapped for tile sharing. Specifically, the tiles within the group are first sorted in ascending order based on the number of empty crossbars (Line 2). This generates a sorted tile list. Then, a two-pointer approach is employed, traversing the list from both ends towards the list center. When the sum of empty crossbars in the two pointed tiles is greater than the total number of crossbars within a tile, the tiles pointed by the two pointers are combined. The empty numbers of crossbars on the two pointers are updated, and the tail pointer is moved one step towards the list center (Lines 8-12). Otherwise, the head pointer moves one step towards the list center (Lines 13-14). This process is repeated until the two pointers meet and the combination results are returned. The algorithm exhibits a time complexity of $O(N)$, where N corresponds to the length of the list.

Algorithm 1 The tile-shared remapping algorithm

```
Require:
        XBNum: the number of crossbars integrated in one tile;
        tileList: a list of tile in one group;
 1: combMap ← empty map
 2: tileList.sort(ascending=True)
 3: head ← 0
 4: tail ← lengthOf(tileList) − 1
 5: while head < tail do
 6:     hTile ← tileList[head]
 7:     tTile ← tileList[tail]
 8:     if hTile.emptyXBNum + tTile.emptyXBNum ≥ XBNum then
 9:         hTile.emptyXBNum ←
                hTile.emptyXBNum + tTile.emptyXBNum − XBNum
10:         tTile.emptyXBNum ← 0
11:         combMap[hTile.ID].append(tTile.ID)
12:         tail ← tail − 1
13:     else
14:         head ← head + 1
15:     end if
16: end while
        return combMap
```

4 EVALUATION

4.1 Experiment Setup

Experiment platform. We implement the AutoHet based on a ReRAM simulator, MNSIM [31], as commercial ReRAMs are currently unavailable to us. We use the MNSIM because of its simplicity and efficiency. We adjust the crossbar size and the number of relevant modules (e.g., DACs, ADCs) in each tile to build different types of tiles. Each bank contains 256×256 tiles while each tile contains four PEs by default. The weights of DNN models are quantized to 8 bits, so we group eight crossbars in each PE to represent one weight data. The precision of memristor cells and DACs is set to 1-bit. We set the ADC revolution to 10-bit to support crossbars of all heterogeneous sizes. We modify the metric calculation module of the hardware model to calculate the system metrics (e.g., RUE, crossbar utilization, and energy consumption) based on crossbar sizes and the number of PCs. All other configurations remain at their default values.

Models and datasets. To evaluate the performance of AutoHet, we select AlexNet [13], VGG16 [11], and ResNet152 [6] as workloads. We also choose three representative datasets. The MNIST [14] dataset consists of 70,000 grayscale $28 \times 28 \times 1$ images. The CIFAR10 [12] dataset comprises 60,000 labeled $32 \times 32 \times 3$ images. The ImageNet [3] dataset has 1.4 million color images with various sizes. We conduct DNN inference of different-scale models with their corresponding datasets (i.e., AlexNet on MNIST, VGG16 on CIFAR-10, and ResNet152 on ImageNet).

Baselines. We compare AutoHet to the existing homogeneous ReRAM-based accelerators. For a comprehensive comparison, we select five commonly used square crossbar sizes to form five homogeneous accelerators as our baselines. Each baseline only chooses one of the five square sizes (i.e., 32×32, 64×64, 128×128, 256×256, and 512×512) for all the DNN layers. For the heterogeneous accelerator, AutoHet automatically chooses one of the five crossbar sizes (i.e., 32×32, 36×32, 72×64, 288×256, and 576×512) for each DNN layer.

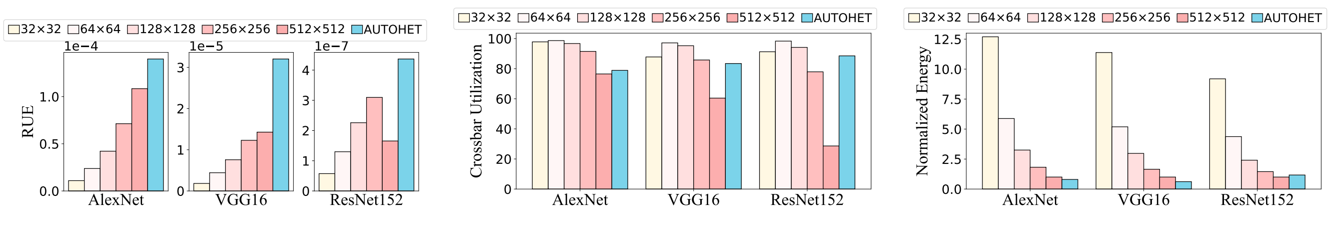
4.2 Overall Performance

Figure 9(a) illustrates the RUE results of different accelerators for the three DNN models. The RUE value is a comprehensive indicator, which is defined as the ratio of the crossbar utilization (U) to the energy consumption (E), as described in §2.2. The larger the RUE value, the better the system. We have two observations.

First, AutoHet consistently exhibits the highest RUE values compared to the homogeneous accelerators. Specifically, its average RUE value is 5.1× higher than those of the homogeneous accelerators across the three DNN models. This demonstrates AutoHet's capability to simultaneously maximize crossbar utilization and energy efficiency. AutoHet performs the best because it uses heterogeneous crossbars, whereas the counterparts utilize homogeneous ones. Furthermore, it considers both metrics through the RL-based search approach, while existing accelerators do not. Additionally, the proposed rectangle crossbar shape and tile-shared crossbar allocation scheme in AutoHet also improve resource utilization, contributing to the RUE values.

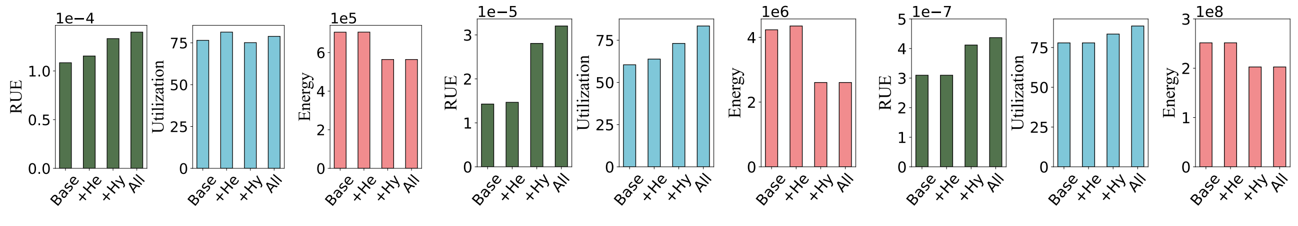
Second, AutoHet demonstrates various speedups on different DNN models. Specifically, it outperforms the best-performing homogeneous accelerator by 1.3×, 2.2×, and 1.4× for AlexNet, VGG16, and ResNet152, respectively. This is because different DNN models have various kernel structures and the number of layers, resulting in varying improvement space that AutoHet can enhance.

Figure 9(b) and 9(c) shows the crossbar utilization and energy consumption of all the accelerators. For better presentation, we normalize the lowest energy consumption of the homogeneous accelerators to one for each DNN model. We observe that although AutoHet sometimes has lower utilization than certain homogeneous accelerators, its energy consumption is significantly reduced. Using the VGG16 model as an example, AutoHet exhibits a slightly lower utilization by 14%, compared to the homogeneous accelerator with 64×64 crossbars, but its energy consumption is reduced by 8.4×. This explains why AutoHet achieves the highest RUE values.



(a) The RUE values (b) Crossbar utilization (c) Energy consumption

Figure 9: The overall performance of different accelerators for the three DNN models.



(a) AlexNet (b) VGG16 (c) ResNet152

Figure 10: The performance of individual techniques in AutoHet for three DNN models.

4.3 Impact of Individual Techniques

Figure 10 shows the impact of each optimization in AutoHet on the RUE, crossbar utilization, and energy consumption. We gradually enable each technique one by one. For simplicity, we use SXB and RXB to denote square crossbar and rectangle crossbar, respectively. SXBs include the five crossbar sizes from the baselines, while RXBs include 36×32, 72×64, 144×128, 288×256, and 576×512. Base means the SXB-based homogeneous accelerator with the highest RUE. For example, for VGG16 on CIFAR-10, it refers to the homogeneous accelerator with the 512×512 SXBs. +He is the accelerator where the RL agent only uses heterogeneous SXBs for each DNN layer. +Hy denotes the version that utilizes both SXBs and RXBs (i.e., 32×32, 36×32, 72×64, 288×256, and 576×512) to construct the accelerator. All extends +Hy to support the tile-shared allocation scheme, which means all optimizations are enabled. We have the following two observations.

First, each optimization applied to the previous version can improve or maintain the RUE, crossbar utilization, and energy consumption. For example, for VGG16, +He achieves a 1.1× increase in RUE compared to Base. This is because +He can choose various suitable SXBs for different DNN layers. When both SXBs and RXBs are used, +Hy achieves 1.9× higher RUE than +He. This is because +Hy improves 9.3% crossbar utilization and reduces 1.7× energy consumption. It demonstrates the advantages of setting the height of the crossbar as a multiple of 9 rather than using the side length as a power of 2 for typical 3×3 kernels. When the tile-shared allocation scheme is enabled, All further increases 1.14× RUE because the scheme can make the empty crossbars in one tile be utilized by other DNN layers. We have similar observations for other metrics (i.e., crossbar utilization and energy consumption) and models (i.e., AlexNet and ResNet152).

Second, the proposed optimizations achieve various improvements for different metrics. For example, +Hy obtains more significant improvement in energy reduction, whereas All achieves more noticeable growth in crossbar utilization. +Hy contributes more in energy reduction because using RXBs makes the crossbars more suitable for the weight matrices, thereby efficiently reducing the number of the most energy-consuming ADCs. For All, its tile-shared allocation scheme allows multiple DNN layers to be mapped onto the same tile, thereby the crossbar utilization can be efficiently boosted.

Table 3: The crossbar size for each layer of VGG16 on MNIST.

Layer	L1	L2-L11	L12-L14	L15	L16
Base	512x512	512x512	512x512	512x512	512x512
+He	512x512	512x512	256x256	512x512	256x256
+Hy	288x256	576x512	576x512	576x512	576x512

Table 4: The number of occupied tiles for different models.

Comparisons	AlexNet	VGG16	ResNet152
+Hy	33	30	246
All	31	27	232

Table 3 shows the assigned crossbar sizes for individual DNN layers in Base, +He, and +Hy. Because of the space limitation, we only show the results of VGG16. Base uses a 512×512 crossbar size for all layers, whereas +He adjusts the crossbar sizes of L12-14 and L16 to 256×256 and keeps 512×512 for the remaining layers. This demonstrates that the RL agent can automatically select appropriate crossbar sizes for different DNN layers, considering their computing characteristics. When RXBs are utilized as heterogeneous crossbar candidates, the 288×256 crossbar is assigned for L1, and the 576×512 is assigned for the remaining layers. This demonstrates that RXBs are more suitable than SXBs in improving utilization and reducing energy consumption for all DNN layers in VGG16. Table 4 illustrates the total number of occupied tiles in +Hy and All. Compared to +Hy, All reduces the number of occupied tiles by 6.1%, 10%, and 5.7% for AlexNet, VGG16, and ResNet152, respectively. This result confirms that the tile-shared allocation scheme can indeed reduce crossbar wastage, thus improving resource utilization.

4.4 Sensitivity Analysis

To show the performance of AutoHet under various scenarios, we evaluate it by adjusting the ratio of SXBs and RXBs, the number of crossbar candidates, and the number of PEs in each tile. Although

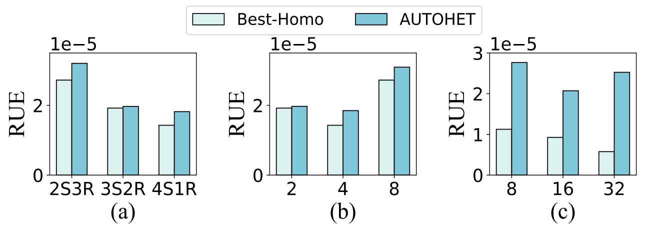


Figure 11: The RUE evaluation of AutoHet under various scenarios: (a) various ratios of SXBs to RXBs, (b) various numbers of crossbar candidates, and (c) various numbers of PEs in each tile.

there are numerous homogenous accelerators available for comparison, we choose the one with the highest RUE value (denoted as Best-Homo) to simplify the presentation. Due to space limitations, we only demonstrate the results for VGG16; however, similar results are observed for other models.

Various ratios of SXBs to RXBs. Figure 11(a) shows the RUE values of AutoHet and Best-Homo with various ratios of SXBs to RXBs. We maintain the total number of crossbar candidates as five and choose a SXBs and bRXBs (i.e., aSbR) from the 10 crossbar sizes mentioned in §4.3. Figure 11(a) shows that AutoHet consistently outperforms Best-Homo: the RUE value is increased from 1.03× to 1.27×. Furthermore, we observe that more RXBs can achieve larger RUE values, demonstrating the effectiveness of the RXBs.

Various numbers of crossbar candidates. We vary the number of crossbar sizes and choose 2, 4, and 8 candidates from SXBs and RXBs. The results are shown in Figure 11(b). It illustrates that AutoHet outperforms Best-Homo by 1.15× on average, regardless of the number of heterogeneous crossbar candidates. Furthermore, we note that AutoHet demonstrates greater RUE improvements compared to Best-Homo when employing a higher number of heterogeneous candidates. This is attributed to the increased diversity of candidates, enabling the RL agent to choose more suitable crossbar sizes for individual DNN layers.

Various numbers of PEs in each tile. We maintain the crossbar candidates fixed but change the number of PEs in each tile from 8 to 32. Figure 11(c) highlights that AutoHet achieves higher RUE values than Best-Homo: it is improved by 2.24× to 4.38×. This shows the generality and robustness of the AutoHet for various underlying hardware configurations.

4.5 Discussion

Area and latency. Table 5 shows the area occupancy and inference latency of AutoHet and other accelerators. Due to space limitation, we only show the results of VGG16; other models have similar observations. We find that AutoHet has the minimum area consumption. Specifically, AutoHet reduces 92% area consumption compared to the best-performing accelerator with homogeneous SXBs (i.e., 512×512). This is because the heterogeneous architecture with hybrid shapes of crossbars and the tile-shared scheme improve the crossbar utilization, thereby reducing the total number of used crossbars and PCs. Furthermore, we observe that the inference latency of AutoHet does not exhibit a significant increase compared to the homogeneous counterparts. Specifically, AutoHet incurs

Table 5: The area occupancy and inference latency of of AutoHet and other accelerators.

Accelerators	SXB32	SXB64	SXB128	SXB256	SXB512	**AutoHet**
Area (μm^2)	2.29E+10	1.02E+10	5.31E+09	3.03E+09	2.12E+09	1.82E+09
Latency (ns)	2.26E+06	2.94E+06	2.96E+06	2.71E+06	2.73E+06	2.34E+06

only 3.2% higher latency than the best-performing 32 × 32 homogeneous accelerator. Compared to other accelerators, AutoHet achieves a latency reduction of 1.2× on average.

Search time of RL. The RL training is an offline search process. The search time includes the time spent by the RL agent to make decisions and the time to wait for rewards from the simulator. Specifically, for VGG16, the 300-round search time is 49.2 minutes. This overhead is acceptable for a contemporary server, as the RL training is executed once but the decision result is used many times. Moreover, we observe that 97% of the search time is spent on the simulator to generate the feedbacks. With the emergence of the real ReRAM-based hardware in the near future, we can generate the feedbacks from the hardware, which is much faster than the simulator. Therefore, the search time on a real hardware platform will be significantly reduced.

Applicability to different domains. In this paper, we verify the effectiveness of AutoHet on DNN models. While, the idea of adopting a heterogeneous architecture based on the characteristics of each part of the model is generic. We believe the idea of heterogeneous ReRAM-based accelerators is effective for other different artificial intelligence domains, such as large language models [4].

5 RELATED WORK

Chip-level heterogeneous architectures. Prior works propose chip-level heterogeneous architectures to accelerate applications. Liu *et al.* [15] proposed a heterogeneous PIM system that combines both fixed-function arithmetic units and programmable cores on the logic layer of a 3D die-stacked memory. Hetraph [8] facilitates energy-efficient graph processing by combining memristor-based analog computing units (for high-parallelism computing) and CMOS-based digital computing cores (for efficient computing). Joardar *et al.* proposed two heterogeneous architectures, REGENT [10] for CNNs and GRAMARCH [9] for DNNs, both of which integrate ReRAM arrays with GPU cores. However, they cannot solve the crossbar wastage in ReRAM, resulting in sub-optimal crossbar utilization.

Crossbar-level heterogeneous architectures. Several studies explore crossbar-level heterogeneous approaches. ReGraphX [1] uses 128×128 crossbars for vertex-computation layers and 8×8 crossbars for edge-computation layers in graph neural network training. REREC [26] adopts 16×16 crossbars to perform inner-product computation and 128×128 crossbars for multi-layer perceptron inference in recommendation systems. However, both of them are not designed for DNN models. Zhu *et al.* [29] proposed to map individual CONV layers onto three SXBs through a greedy algorithm. However, it focuses more on crossbar utilization while AutoHet aims at both crossbar utilization and energy consumption through the RL-based method. Furthermore, AutoHet leverages RXBS and tile-shared allocation to further boost resource utilization.

AutoML methods for DNNs on ReRAMs. Some efforts leverage AutoML methods to improve the behaviors of DNN models on

ReRAM-based accelerators, such as pruning [27], quantization [25], and mapping [18]. However, they pay more attention to the model optimization instead of the hardware architecture. In contrast, AutoHet uses an RL-based method to design efficient ReRAM-based accelerator architecture for DNN models.

6 CONCLUSION

In this paper, we propose AutoHet, an automated heterogeneous ReRAM-based accelerator for DNN models to maximize crossbar utilization while minimize energy consumption. Given the characteristics of each layer in the DNN, AutoHet automatically selects appropriate heterogeneous crossbars for each layer using reinforcement learning. Additionally, we introduce hybrid crossbar shapes (i.e., square and rectangle crossbars) to further enhance the matching between the weight matrices and crossbars. Finally, we propose the tile-shared allocation scheme to improve utilization by allowing multiple DNN layers to be mapped onto the same tile. Experimental results demonstrate that AutoHet effectively improves the crossbar utilization by up to 3.1× while reducing the energy consumption by up to 94.6% compared to existing homogeneous accelerators.

ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China (2023YFB4502100), the Open Project Program of Wuhan National Laboratory for Optoelectronics (2023WNLOKF005), the National Science Foundation of China (62172361), the Major Projects of Zhejiang Province (LD24F020012), the National Key Research and Development Program of China (2021ZD0110700).

REFERENCES

[1] Aqeeb Iqbal Arka, Janardhan Rao Doppa, Partha Pratim Pande, Biresh Kumar Joardar, and Krishnendu Chakrabarty. 2021. ReGraphX: NoC-enabled 3D Heterogeneous ReRAM Architecture for Training Graph Neural Networks. In *Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 1667–1672.
[2] 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. *ACM SIGARCH Computer Architecture News* (2016), 27–39.
[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-scale Hierarchical Image Database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.
[4] Yingqiang Ge, Wenyue Hua, Kai Mei, Juntao Tan, Shuyuan Xu, Zelong Li, Yongfeng Zhang, et al. 2024. OpenAGI: When LLM Meets Domain Experts. *Advances in Neural Information Processing Systems* (2024).
[5] Mehdi Ghasemi, Soroush Heidari, Young Geun Kim, Aaron Lamb, Carole-Jean Wu, and Sarma Vrudhula. 2021. Energy-Efficient Mapping for a Network of DNN Models at the Edge. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*. 25–30.
[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, (CVPR)*. 770–778.
[7] Hung-Hsi Hsu, Tai-Hao Wen, Wei-Hsing Huang, Win-San Khwa, Yun-Chen Lo, Chuan-Jia Jhang, Yu-Hsiang Chin, Yu-Chiao Chen, Chung-Chuan Lo, Ren-Shuo Liu, et al. 2023. A Nonvolatile AI-Edge Processor with SLC–MLC Hybrid ReRAM Compute-in-Memory Macro Using Current-Voltage-Hybrid Readout Scheme. *IEEE Journal of Solid-State Circuits* (2023).
[8] Yu Huang, Long Zheng, Pengcheng Yao, Jieshan Zhao, Xiaofei Liao, Hai Jin, and Jingling Xue. 2020. A Heterogeneous PIM Hardware-software Co-design for Energy-efficient Graph Processing. In *Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 684–695.
[9] Biresh Kumar Joardar, Nitthilan Kannappan Jayakodi, Janardhan Rao Doppa, Hai Li, Partha Pratim Pande, and Krishnendu Chakrabarty. 2020. GRAMARCH: A GPU-ReRAM Based Heterogeneous Architecture for Neural Image Segmentation. In *Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 228–233.
[10] Biresh Kumar Joardar, Bing Li, Janardhan Rao Doppa, Hai Li, Partha Pratim Pande, and Krishnendu Chakrabarty. 2019. REGENT: A Heterogeneous ReRAM/GPU-based Architecture Enabled by NoC for Training CNNs. In *Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 522–527.
[11] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arxiv: 1409.1556* (2014).
[12] Alex Krizhevsky. 2012. Learning Multiple Layers of Features from Tiny Images. *University of Toronto* (2012).
[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification With Deep Convolutional Neural Networks. *Commun. ACM* (2017), 84–90.
[14] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *IEEE* (1998), 2278–2324.
[15] Jiawen Liu, Hengyu Zhao, Matheus A Ogleari, Dong Li, and Jishen Zhao. 2018. Processing-in-memory for Energy-efficient Neural Network Training: A Heterogeneous Approach. In *Proceedings of the 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 655–668.
[16] Sparsh Mittal. 2019. A Survey of ReRAM-Based Architectures for Processing-In-Memory and Neural Networks. *Machine Learning and Knowledge Extraction* (2019), 75–114.
[17] S. S. Mousavi, M. Schukat, and E. Howley. 2018. Deep Reinforcement Learning: An Overview. *Springer, Cham* (2018).
[18] Songyun Qu, Bing Li, Ying Wang, Dawen Xu, Xiandong Zhao, and Lei Zhang. 2020. RaQu: An Automatic High-Utilization CNN Quantization and Mapping Framework for General-Purpose RRAM Accelerator. In *Proceedings of then 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[19] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, 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. *ACM SIGARCH Computer Architecture News* (2016), 14–26.
[20] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*. 387—-395.
[21] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. 2017. Pipelayer: A Pipelined ReRAM-Based Accelerator for Deep Learning. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 541–552.
[22] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S. Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.
[23] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. 2019. Mnasnet: Platform-aware Neural Architecture Search for Mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2820–2828.
[24] Shikhar Tuli and Shreshth Tuli. 2020. AVAC: A Machine Learning Based Adaptive RRAM Variability-aware Controller for Edge Devices. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[25] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. 2019. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 8612–8620.
[26] Yitu Wang, Zhenhua Zhu, Fan Chen, Mingyuan Ma, Guohao Dai, Yu Wang, Hai Li, and Yiran Chen. 2021. Rerec: In-ReRAM Acceleration with Access-Aware Mapping for Personalized Recommendation. In *Proceedings of the 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.
[27] Siling Yang, Weijian Chen, Xuechen Zhang, Shuibing He, Yanlong Yin, and Xian-He Sun. 2021. AUTO-PRUNE: Automated DNN Pruning and Mapping for ReRAM-Based Accelerator. In *Proceedings of the ACM International Conference on Supercomputing (ICS)*. 304–315.
[28] Siling Yang, Shuibing He, Hexiao Duan, Weijian Chen, Xuechen Zhang, Tong Wu, and Yanlong Yin. 2023. APQ: Automated DNN Pruning and Quantization for ReRAM-Based Accelerators. *IEEE Transactions on Parallel and Distributed Systems* (2023).
[29] Zhenhua Zhu, Jilan Lin, Ming Cheng, Lixue Xia, Hanbo Sun, Xiaoming Chen, Yu Wang, and Huazhong Yang. 2018. Mixed Size Crossbar Based RRAM CNN Accelerator with Overlapped Mapping Method. In *Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.
[30] Zhenhua Zhu, Hanbo Sun, Yujun Lin, Guohao Dai, Lixue Xia, Song Han, Yu Wang, and Huazhong Yang. 2019. A Configurable Multi-Precision CNN Computing Framework Based on Single Bit RRAM. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC)*. 1–6.
[31] Z. Zhu, H. Sun, K. Qiu, L. Xia, G. Krishnan, G. Dai, D. Niu, X. Chen, X. S. Hu, and Y. K. Cao. 2020. MNSIM 2.0: A Behavior-Level Modeling Tool for Memristor-based Neuromorphic Computing Systems. In *Proceedings of the GLSVLSI '20: Great Lakes Symposium on VLSI 2020*. 83–88.