Hierarchical-ISA Supporting Row-Wise Operands for Efficient DNN Computation

Zhiwang Huo, Wenzhe Zhao, *Member, IEEE*, Qiwei Dang, Chengyu Ma, Guoming Yang, Gelin Fu, *Member, IEEE*, Tian Xia, *Member, IEEE*, and Pengju Ren, *Member, IEEE*

***Abstract*—Deep neural networks (DNNs) have become a cornerstone in advancing artificial intelligence, but their complexity often leads to inefficient hardware utilization due to varying structure characteristics and excessive memory accesses. Domain-specific architectures (DSAs) offer a solution by optimizing data locality through data stationarity, tiling, and layer fusion, which minimize memory access and energy consumption while boosting performance. However, current approaches lack flexibility for efficient memory management at the appropriate granularity, causing misaligned accesses and decreasing reuse potential for variable-sized tiles. To this end, we propose a hierarchical instruction set architecture (hierarchical-ISA) combining a RISC-V ISA and a flexible CISC-style macro-ISA (mISA). Unlike byte-level RISC-V, mISA employs row-wise tiles as the fundamental operand, enabling efficient data reuse across adjacent iterations as well as residual connections. This mISA approach simplifies DNN programming, enhances data partitioning and manipulation efficiency, and enables a hardware–software co-designed Remapping mechanism that facilitates data reuse without physical data movement. Experiments show 31.8%–72.0% reductions in off-chip memory access across MobileNet, ResNet, Swin Transformer, and MobileViT, along with speedups of 2.9×–7.4× compared to previous DNN accelerators. We also conduct comparisons under the roofline model with NVIDIA RTX A6000 and Intel Core i7-10700K. The results show that our arithmetic intensity (AI) reaches up to 26.0× that of i7-10700K and 22.6× that of A6000.**

***Index Terms*—Deep neural network (DNN), domain-specific architecture (DSA), hardware–software co-design, on-chip memory (OCM).**

I. Introduction

DEEP neural networks (DNNs) have garnered widespread attention for their outstanding performance in artificial intelligence tasks, such as speech recognition [1], [2], image classification [3], [4], and object detection [5], [6]. As computational demands increase, microprocessors have adopted two main trajectories to improve parallel computing performance. One trajectory is latency-oriented design, represented by multicore CPUs, and the other is throughput-oriented, represented by GPUs with massive threads [7]. Although GPUs attempt to exploit on-chip memory (OCM) locality through various thread block and warp scheduling strategies, the diverse types of layers within DNNs exhibit varied localities and computational characteristics, leading to inadequate utilization of hardware resources and limitations in overall efficiency [8].

In comparison to general-purpose processors, domain-specific architectures (DSAs) can aggressively exploit data locality through the flexible utilization of techniques such as data stationarity [8], [9], [10], [11], tiling [12], [13], [14], [15], and layer fusion [16], [17], [18], [19], thereby dramatically reducing memory access, lowering power consumption, and enhancing performance. However, the challenge of exploiting data locality in DSAs becomes more significant. For example, extensive layer fusion across multiple layers creates large overlap regions between successive tiles [16]. These overlaps, especially in shallow layers, offer potential for reuse, but their patterns are often irregular. Managing this reuse complicates the dataflow significantly, because the boundaries of these overlap regions depend critically on numerous factors including tile dimensions ($H \times W \times C \times N$ for height, width, channels, batch size), weight shape, stride, and tile sliding direction [20]. Furthermore, traditional operand management, optimized at the byte level using scalar or vector registers, proves inadequate for the kilobyte-level optimizations required by DNN operations like layer fusion. As a result, memory access misalignment occurs when handling variable-size tiles, forcing costly multicycle cross-address indexing.

The line-buffer depth-first (LBDF) [18] simplifies the overlap regions caching by using line tiles of the feature map (referred to as Row-wise Tiling). Compared to other layer fusion techniques, Row-wise Tiling aligns with tensor characteristics by partitioning feature maps along the row dimension. This approach naturally matches the spatial locality of convolution and minimizes memory requirements within a concise reuse pattern [21]. However, Row-wise Tiling faces two significant limitations: first, inflexible OCM organization and management hinder its effective utilization [19]. Second, Row-wise Tiling proves less efficient in networks with residual blocks. Since inputs are discarded immediately after their first use, such as in LBDF, the residual block inputs—comprising 43% of activations in ResNet20 and 16% in MobileNetV2—must be reloaded, incurring substantial overhead. Consequently, efficient OCM management is crucial not only for maximizing the benefits of tiling and fusion but also specifically for overcoming the long-latency data reuse demands introduced by residual blocks.

Received 24 March 2025; revised 18 August 2025 and 16 November 2025; accepted 23 November 2025. Date of publication 25 November 2025; date of current version 22 June 2026. This work was supported in part by National Natural Science Foundation of China under Grant 62088102, Grant 62302381, and Grant 52441602. This article was recommended by Associate Editor M. Shafique. *(Corresponding author: Wenzhe Zhao.)*

The authors are with the National Key Laboratory of Human-Machine Hybrid Augmented Intelligence, the National Engineering Research Center for Visual Information and Applications, and the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China (e-mail: wenzhe@xjtu.edu.cn).

Digital Object Identifier 10.1109/TCAD.2025.3637162

1937-4151 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

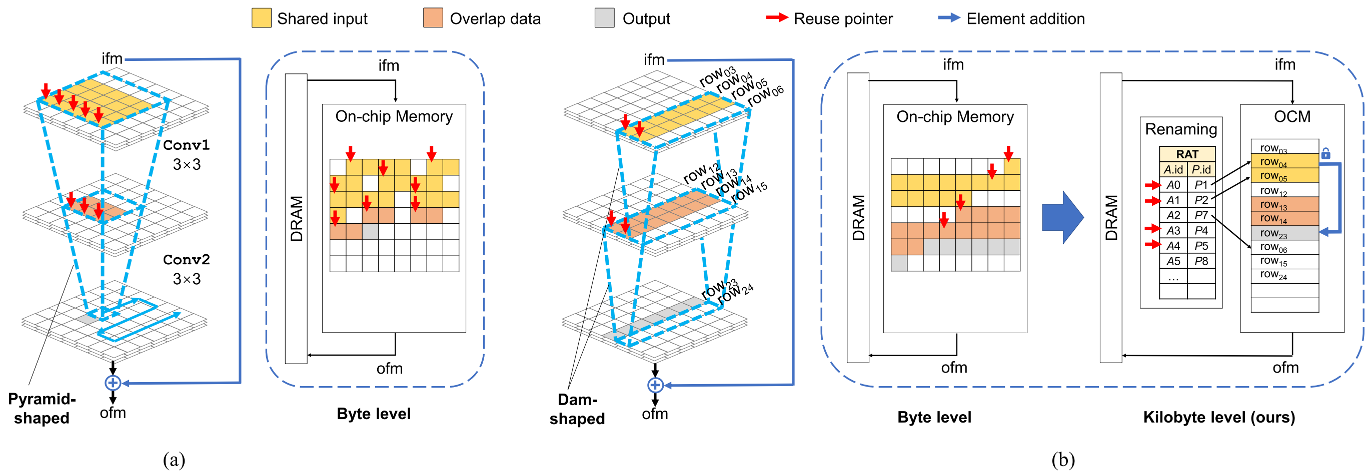


Fig. 1. (a) DF tiling (pyramid-shaped) and (b) Row-wise Tiling (dam-shaped) both enhance data locality by reusing shared inputs and overlap data. However, byte-level OCM management leads to irregular reuse pointers due to severe OCM fragmentation during tile sliding. To address this, we build upon Row-wise Tiling and employ KB-level OCM management. Physical and logical OCM are decoupled through the renaming technique, which simplifies reuse pointer generation. Specifically, each mReg stores one row tile. An mReg is divided into the macro architecture register (mAR, denoted as A) and the macro physical register (mPR, denoted as P). Software scheduling maps ifms/activations to an A.id. Hardware scheduling executes renaming, mapping the A.id to a P.id, where the P.id points to the OCM space that stores the corresponding row tile.

From a programming perspective, DSAs lack the comprehensive and rich ecosystem that CPUs and GPUs offer, resulting in high development investment [22]. Moreover, the deployment of DNNs on DSAs has become increasingly challenging due to the emergence of wider operands, more memory footprint, and sophisticated network models spurred by DNN breakthroughs [23]. For example, the attention mechanism [24] allows the model to weigh the importance of different input elements relative to each other, capturing long-range dependencies more effectively than traditional convolution. The graph neural network [25] performs convolution on the nodes of a graph, taking into account the connections between nodes. The rapid evolution of these convolution operations and network models presents significant challenges to DSA design. DSAs often need to make tradeoffs in programmability to enhance performance for specific applications.

Given the co-design challenges posed by the development of DNNs, there is a pressing need to develop an improved instruction set architecture (ISA) as well as a corresponding programming paradigm that supports Row-wise Tiling, reduces data traffic, and simplifies custom software development. To address this, we propose a hierarchical-ISA framework incorporating a CISC-style macro ISA (mISA). This approach strategically decouples Row-wise Tiling optimization—managed by the mISA scheduler—from operator execution handled by the RISC-V core. Crucially, mISA treats row tiles as fundamental operands, enabling efficient orchestration of row-level dataflow optimizations. Our specific contributions are as follows.

1) A hierarchical-ISA featuring mISA for single-instruction execution of complex operations, with a streamlined programming paradigm that simplifies coding and facilitates operator fusion.
2) Macro registers (mRegs) abstracting row-tile granularity, enabling dynamic task allocation with variable tile sizes, and enabling flexible data placement and recycling within OCM.
3) A Remapping mechanism for reusable data lifecycle management. This mechanism eliminates false dependencies and pipeline bubbles by enabling logical mapping between mRegs—facilitating data reuse without physical movement—making it particularly effective for residual connections.

The remainder of this article is organized as follows. Section II introduces the basic concepts of layer fusion and Row-wise Tiling, and the performance evaluation model. Section III describes the mISA-based programming paradigm. Section IV presents the mRegs-based hardware Remapping mechanism. Experimental results are given in Section V. Related works are discussed in Section VI. Section VII concludes this article.

II. Background and Motivation

A. Layer Fusion and Row-Wise Tiling

Exploiting data locality becomes necessary in DNN inference to reduce memory access and latency [26]. Fig. 1(a) demonstrates this using a ResNet18 residual block [27], featuring two 3 × 3 convolutions (`Conv1` and `Conv2`) where the results, after cascading, undergo element-wise addition (`Ele+`) with the input. In such blocks,[1] the output from the preceding layer is the input to the following layer. To capitalize on this interlayer locality, software can merge multiple layers into a singular entity, thus only transferring input feature maps (ifms) and output feature maps (ofms) to/from DRAM.

Fig. 1(a) shows a depth-first (DF) tiling [16] based on the byte level, forming a pyramid-shaped region (dashed outline) across multiple layers. Once the ifm of a pyramid is loaded, the other layers (intermediate activations) can be calculated without any off-chip access. After computing this pyramid, it shifts by one stride, requiring only new columns of the ifm tile for the next output. Overlap pyramids (orange region) share intermediate data for adjacent ofm elements, thus caching the

[1] In this example, all filters are applied with padding and stride = 1.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

overlap region saves memory footprint theoretically. However, managing these overlaps demands sophisticated control based on input–output relations and computational patterns, varying dramatically across layers [20]. The red pointers in Fig. 1(a) indicate the irregular positions of reusable data in OCM. To address the challenges of indexing complexity and fragmentation, Alwani et al. [16] copied the reusable data from the shared buffer to a dedicated buffer. This copy function, in turn, requires a three-level loop to index the data.

As shown on the left of Fig. 1(b), LBDF [18] optimizes the data reuse in the row direction of the feature map. The process of Row-wise Tiling creates a dam-shaped region (dashed outline), which spans multiple layers. After buffering several rows of feature maps (yellow and orange areas), the layer can start computing new outputs as soon as it receives input from the previous layer. To maximize the utilization of OCM, new data continuously overwrites the old data. Due to varying row sizes and space requirements across different layers, the complexity of calculating reuse pointers escalates as the number of fused layers grows. Nevertheless, Shi et al. [21] highlighted that, in some cases, recalculating overlap data can be more efficient than caching overlap data in LBDF, particularly when the intermediate layers have shapes larger than the input layers, such as inverted bottlenecks of deeper models. This is because relying entirely on hardware to perform flexible OCM allocation and management would incur nonnegligible overhead.

To address these challenges, as illustrated in the right of Fig. 1(b), we adopt a register-based management approach to dynamically partition the OCM into variable-sized mRegs. These mRegs support operands at a row-wise tile granularity to accommodate different network structures and layers. By employing the register renaming technique, mReg is classified into macro architecture register (mAR, labeled A) and macro physical registers (mPR, labeled P). Therefore, programmers can specify row tile location through mAR, and the actual physical location is indexed by mPR. Meanwhile, the renaming mechanism guarantees the correct relationship between mAR and mPR.

Furthermore, the initial inputs of residual blocks account for a significant portion (21.5% in our ResNet18 experiments using LBDF) of total activation volumes; thus, these inputs should not be discarded immediately after first use. However, the structural characteristics of residual connections spanning multiple layers create long latency intervals between their production and subsequent reuse. Without an effective OCM allocation and recycling mechanism, these long-interval reused data force costly reloads, incurring substantial overhead. Thus, we introduce a data Lifecycle Analysis mechanism for residual data management. Attributed to the regularity of the reuse pattern in the overlap regions, Row-wise Tiling concurrently streamlines the control of OCM operations.

B. Guidance From the Roofline Model

The roofline model is a framework used to analyze and optimize the performance of variations of architectures, helping developers identify performance bottlenecks [28]. This model categorizes the limiting factors of performance into compute-bound and memory-bound, visually represented by a 2-D graph. The core concept of the roofline model is arithmetic intensity (AI), which refers to the number of operations per byte of memory accessed. A high AI indicates that a program's performance is more dependent on hardware computational power, while a low AI suggests that the program is data-starved and requires higher memory bandwidth.

Google [29] has shown that neural networks are more prone to being memory-bound. Therefore, designers try their best to use a variety of parallel computing techniques to perform more calculations on resident data. The upper limit of AI is mainly determined by the potential for data reuse. The reuse of feature maps (R_f) and weights (R_w) is quantified using the following formulas:

$$R_\mathrm{f} = \left(\left\lfloor \frac{K-1}{S} \right\rfloor + 1\right)^2 \times C_\mathrm{out} + RC \tag{1}$$

$$R_\mathrm{w} = \left(\left\lfloor \frac{H+2P-K}{S} \right\rfloor + 1\right) \times \left(\left\lfloor \frac{W+2P-K}{S} \right\rfloor + 1\right) \tag{2}$$

where K, S, C_out, P, and RC represent the kernel size, stride, output channel, padding, and residual connection numbers, respectively.

Formula (1) indicates that increasing kernel size, output channels, and residual connections can achieve larger R_f, which, in turn, elevates the upper limit of AI. Optimizing feature map reuse efficiency R_f is critical for achieving high AI, because weight stationary [10] keeps weights in OCM throughout inference, enabling maximal R_w realization. Given INT8 precision (1 byte) with all intermediate data held on OCM, the theoretical upper bound of AI can be calculated as follows:

$$\begin{aligned}\mathrm{AI}_\mathrm{max} &= \frac{\text{Operands}}{\text{Data Movement}_\text{optimized}} \\ &= \frac{\text{Operands}}{\text{Input}_\text{model} + \text{Output}_\text{model} + \text{Param}}.\end{aligned} \tag{3}$$

Here, C_in represents the input channel. Shortcut Fusion [19] establishes a baseline Data $\text{Movement}_\text{baseline}$ where inputs/outputs/parameters transfer from/to off-chip memory exactly once per layer. Actual data movement deviates from this baseline depending on OCM management strategies. Thus, actual AI can be represented as follows:

$$\begin{aligned}\mathrm{AI}_\mathrm{act} &= \frac{\text{Operands}}{\sum\left(\text{Input}_\text{layer} + \text{Output}_\text{layer}\right) + \text{Param}} \\ &\quad \times \frac{1}{(1 - \text{Off-chip Reduction})}.\end{aligned} \tag{4}$$

Advanced locality optimization can significantly enhance the off-chip reduction, bringing AI_act closer to the theoretical maximum AI_max. However, translating this potential into practical implementation faces a critical challenge: inherent irregular memory access patterns associated with layer fusion. Sections III and IV will detail how it systematically overcomes this barrier through Row-wise Tiling combined with Remapping and Lifecycle Analysis.

III. Programming Paradigm

A. Tile mReg

For the DNN accelerator, the OCM is always organized as a scratchpad; in other words, data management is explicitly

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

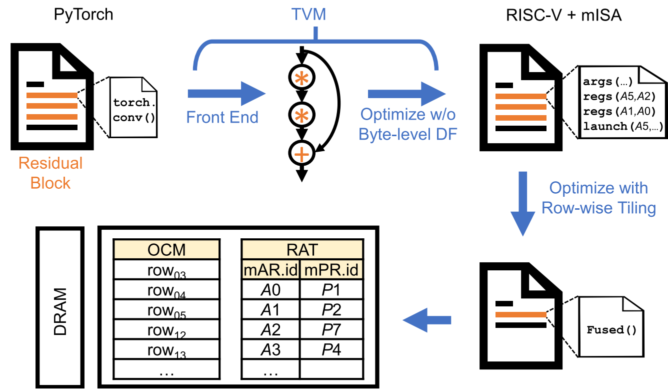


Fig. 2. Residual block compilation with Row-wise Tiling. Progressive lowering transforms PyTorch tensors to logic mRegs (mAR.id) then physical mRegs (mPR.id) via Remapping.

controlled by the programmer. To better support Row-wise Tiling, we manipulate OCM units into variable-sized mRegs that are explicitly specified in the mISA and further use the register renaming scheme to manage them accordingly. The data partitioning strategy involves dividing both ifms and activations into row tiles and storing them in mRegs. This scheme allows software to exploit data locality through explicit programming, without needing to worry about register conflicts. The variability in tile sizes, crucial for different tasks, introduces memory management challenges such as utilization and overwriting, as discussed in SmartShuttle [30]. In our configuration, we use 6 bits to specify the index of 64 mRegs. Notice that we can also increase the number of mRegs to accommodate different computational needs.

DNNs leverage tiling and loop unrolling to process batched inputs efficiently. To align with both the typical page size and the fundamental block RAM of FPGAs, we set a base unit of 4 kB for mReg. This design benefits most models in our experiments. Crucially, our design allows a single mReg to contain 1–8 nonadjacent 4 kB units to accommodate larger rows, thereby mitigating OCM fragmentation. To validate this approach, we implement our design on FPGA prototypes to evaluate the proposed method's effectiveness.

B. Software Organization Based on mISA and Kernel Function

Our accelerator scheduler employs the mISA, while processing cores utilize the RISC-V ISA. The key advantage of mISA lies in its capacity to execute complex operations in a single instruction or a combination of basic instructions. This capability empowers programmers to achieve more with fewer instructions, thereby simplifying the programming and debugging processes. Complementing this, each processing core implements a RISC-V ISA specifically tailored for distinct operators. This hierarchical-ISA architecture achieves dual objectives: 1) enabling developers to focus on custom operator efficiency through streamlined microarchitecture programming and 2) maintaining compatibility with emerging operator standards via the extensible RISC-V ecosystem. Fig. 2 illustrates the compilation workflow using ResNet18's residual block as an implementation case. The diagram details how computational graphs are lowered through TVM [31] and mISA passes, where operators are fused according to our Row-wise Tiling strategy, in contrast to conventional byte-level DF Tiling.

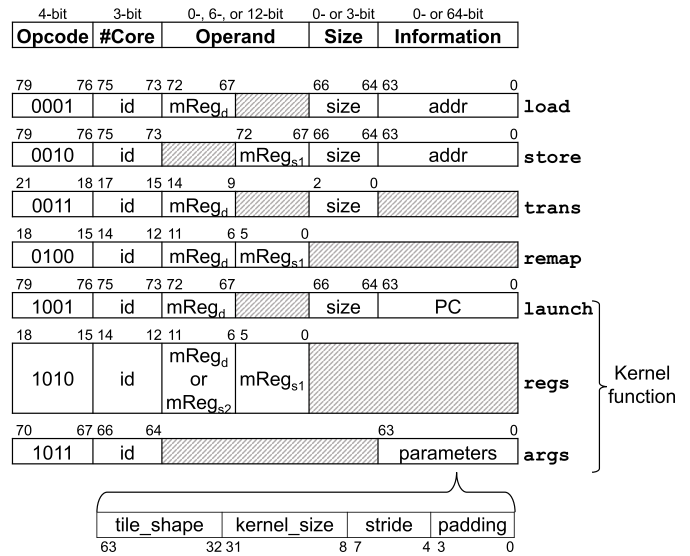


Fig. 3. Macro instruction set format.

As shown in Fig. 3, the proposed mISA consists of five fields. To maintain the extensibility of the instruction set, the "Opcode" field is set to 4 bits for further extension. The "#Core" field specifies the particular core upon which the current instruction operates. By default, 3 bits are used to support up to 8 cores, each being a general processing unit supporting RISC-V. The "Operand" field contains up to two mRegs, which can be either a destination mReg ($mReg_d$) or a source mReg ($mReg_s$),[2] with each mReg represented by 6 bits. The "Size" field indicates the OCM size allocated for a mReg, with the current design supporting a dynamic range from 1 to 8 (4 to 32 kB, respectively).[3] For storage requirements that exceed this range, the issue can be addressed by adjusting the architecture to increase the "Size" or by further partitioning the data. The "Information" field encompasses arguments associated with the specific macro instruction.

For clarity, Fig. 3 aligns instruction fields across all seven macro instructions we introduce. `load`/`store`: Transfer data between DRAM and $mReg_d$ or $mReg_s$. `trans`: Transfer data between cores, thus facilitating task pipelining across cores with deeper layer fusion. `remap`: Define explicit re-identification relationship between mRegs. The `launch`, `regs`, and `args` constructs basic kernel functions, where `args` configures arguments for kernel function (e.g., tile shape, kernel size, stride, and padding as shown in Code 1 for `Conv1`). `regs` allocates the mRegs necessary for the kernel function. After that, `launch` instruction initiates computation specified by the program counter (PC), which points to RISC-V-based control flow specifying operator execution. These PC values are statically generated during the compilation phase. Finally, save the result to the destination mReg ($mReg_d$). Additionally, `regs` can also configure the source mReg of `trans` instruction.

Our mISA-based programming paradigm enables kernel launch through a streamlined workflow: host code initiates

[2] By default, in `regs` instruction, the data in $mReg_{s1}$ serves as the destination mReg, while the remaining $mReg_s$ act as source mReg.

[3] For clarity, the number in the "Size" field within the instruction is the actual memory size in the following description.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

Code 1 Operator Library Based on the Kernel Function

```
1:  function Conv1(#Core, A5, [A0, A1, A2], tile_shape, ...):
2:      args(#Core, tile_shape, ...)
3:      regs(#Core, A5, A2)
4:      regs(#Core, A1, A0)
5:      launch(#Core, A5, size, PC.conv)
6:  function Conv2(#Core, ...):
7:      ...
8:  function Ele+(#Core, A6, [A0, A6], tile_shape, ...):
9:      args(#Core, tile_shape, ...)
10:     regs(#Core, A6, A0)
11:     launch(#Core, A6, Size, PC.ele)
12: function Trans(#Core1, A2, #Core0, A6, size):
13:     regs(#Core0, A6)            // Set transfer source mReg
14:     trans(#Core1, A2, size)
```

Code 2 Fused Execution of Two Residual Blocks

```
Initialization:
▷ Preload weights/biases
▷ Generate synchronization counters to coordinate pipeline execution between cascaded residual blocks
▷ Suppose 8KB row tile size

1:  function Fused(tile0_shape, ..., size, ..., ifm_addr, ...):
    ▷ The first residual block
2:  for row = 0 to tile0_shape.height - 1 do
3:      load(#0, A0, 8KB, ifm_addr+row×8K)
4:      load(#0, A1, 8KB, ifm_addr+(row+1)×8K)          ❶
5:      load(#0, A2, 8KB, ifm_addr+(row+2)×8K)          ❷
6:      remap(#0, A3, A4)                               ❸
7:      remap(#0, A4, A5)                               ❹
8:      Conv1(#0, A5, [A0, A1, A2], tile0_shape, ...)
9:      Conv2(#0, A6, [A3, A4, A5], tile1_shape, ...)
10:     Ele+(#0, A6, [A0, A6], tile0_shape, ...)
11:     Trans(#1, A2, #0, A6, 8KB)
    ▷ Transfer data from A6 in #0 to A2 in #1
    ▷ The second residual block
12:     remap(#1, A0, A1)
13:     remap(#1, A1, A2)
14:     remap(#1, A3, A4)
15:     remap(#1, A4, A5)
16:     Conv1(#1, A5, [A0, A1, A2], tile2_shape, ...)
17:     Conv2(#1, A6, [A3, A4, A5], tile3_shape, ...)
18:     Ele+(#1, A6, [A0, A6], tile2_shape, ...)
19:     store(#1, A6, 8KB, ofm_addr+row×8K)
    ▷ Only single load/store executed in Fused function
20: end for
```

kernels via standard *C* interfaces to configure operational parameters, automatically unrolls for loops, and ultimately executes functionality through macro instruction bundles. Our operator library supports function calls within kernel functions at runtime. This allows for the creation of distinct RISC-V style operators (e.g., graph pooling and attention mechanism) that can be seamlessly assembled with the current operator library. As shown in Code 2, it is feasible to concatenate `Conv1`, `Conv2`, and `Ele+` consecutively within a residual block and then concatenate two residual blocks within a Fused function. This enhances the composability of the operator library, facilitating seamless integration into deep learning software stacks like TVM. The code is available at https://github.com/zhiwang3/misa.git

IV. Hardware and Software Co-Design

This section provides an explanation of how the hardware Remapping mechanism allocates OCM space for a variable-sized tile, elucidating the process of determining the lifecycle of reusable data and delineating the protocol for releasing storage resources. The Remapping mechanism includes the Input List, Register Alias Table (RAT), and Address Mapping Table.

A. Data Reuse Based on Remapping

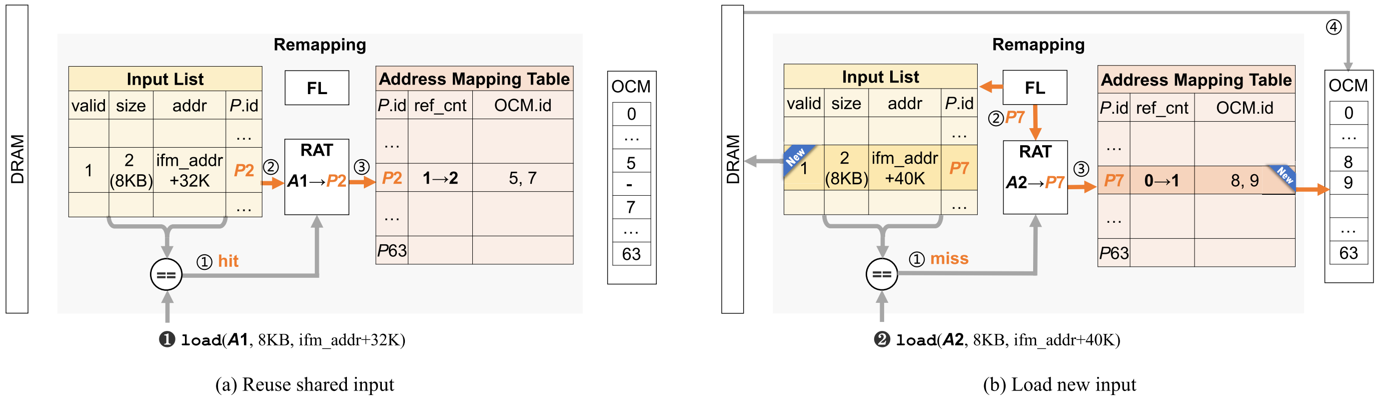
The Input List tracks data already loaded into OCM, recording valid status, address, size, and associated *P*.id. Complementing this, the Address Mapping Table records specific OCM location (OCM.id) allocated to each *P*.id, where a nonzero reference counter (ref_cnt) indicates a valid entry. To ascertain whether the required data has been loaded into OCM, a comparison is conducted between the address and size specified within the `load` instruction and the corresponding address and size in the Input List. Reusable data falls into shared input and overlap intermediate activation. The shared input has the following.

1) *Hit Handling (❶ `load(A1,...)`):* As shown in Fig. 4(a), a hit for a `load` instruction indicates the required row data resides in OCM. The associated Input List entry provides the *P*.id (*P*2), and the RAT redirects *A*1 to *P*2.
2) *Miss Handling (❷ `load(A2,...)`):* As shown in Fig. 4(b), ① a miss occurs when loading new input. If insufficient OCM space or FL entries exist, back-pressure stalls the instruction until resources are free. Otherwise, the allocation protocol proceeds: ② An unallocated *P*.id (e.g., *P*7) is obtained from the FL, and sufficient OCM space (e.g., 8 kB at OCM.id 8–9) is allocated. ③ The RAT, FL, Address Mapping Table, and Input List are updated. *A*2 is mapped to *P*7 in the RAT, *P*7 is marked allocated in the FL, the Address Mapping Table links *P*7 to the OCM.id(s), and the Input List caches *A*2's address and size under *P*7. ④ Data is loaded from DRAM into the allocated OCM space.

For overlap intermediate activation: The `remap` instruction reuses overlapping data across iterations, as shown in Fig. 5. In Code 2, *A*3 and *A*4 in iteration $(i+1)$ correspond to *A*4 and *A*5 in iteration i. For `remap` (e.g., ❸ `remap`(*A*3, *A*4)): 1) the *P*.id (*P*4) referenced by the source operand (*A*4) is retrieved from the RAT and 2) the destination operand (*A*3) is directly redirected to this same *P*.id (*P*4) in the RAT. The same applies to instruction ❹. After that, execute the subsequent instructions. Unlike rotating register allocation [32], which relies heavily on compiler support, our approach achieves register reuse automatically through hardware mechanisms, simplifying programming for data reuse across loops.

In the current design, the 64-entry Address Mapping Table dynamically allocates OCM space in 1–8 units of 4 kB each, supporting up to 2048 kB logically. All valid and ref_cnt

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.



(a) Reuse shared input

(b) Load new input

Fig. 4. (a) Load hit case for ❶: ① a hit occurs if the request address and size within the load instruction match an existing entry in the Input List. ② obtain the P.id($P2$) corresponding to the requested A.id($A1$) and update the RAT. ③ increment the ref_cnt for $P2$ in the Address Mapping Table. ref_cnt = 2 indicates that the operand in $P2$ will be used by two instructions. (b) Load miss case for ❷: ① a miss occurs if the request data does not match any entry in the Input List. ② identify an idle P.id (e.g., $P7$) from the FL and an available OCM.id (e.g., 8 and 9) from OCM. ③ upon successful allocation, record the mapping between the allocated P.id and OCM.id in the Address Mapping Table. ④ transfer the requested 8 kB data from DRAM to the allocated OCM location.

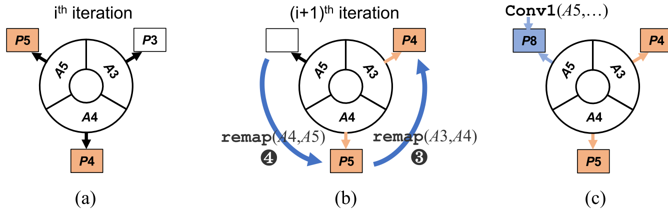


Fig. 5. (a) Intermediate data generated in the ith iteration are stored in different P.id entries, and the overlap data in $P4$ and $P5$ can be reused in the $(i+1)$th iteration. (b) In the $(i+1)$th iteration, the `remap` is used to redirect $P4$ and $P5$ to $A3$ and $A4$, respectively. (c) `Conv1`($A5, \ldots$) will allocate a new P.id for the destination $A5$. It can either select $P3$, which is just released after ith iteration, or reallocate a new P.id, like $P8$ shown in the figure.

signals in the Input List, FL, RAT, and Address Mapping Table are initialized to 0. Valid signals are set to 1 upon entry assignment. When the core switches execution tasks, all valid signals are reset to 0 for proper state management. Successfully processed `load` or `remap` instructions dispatch the macro instruction to the issue queue, signaling readiness for execution.

B. Residual Connection Based on Lifecycle Analysis

To facilitate hardware analysis of data lifecycles, a reference counter (ref_cnt) is implemented within the Address Mapping Table. The ref_cnt signifies the count of instructions utilizing the data in P.id. Thus, in scenarios ❷ the ref_cnt corresponding to $P7$ would be incremented by 1, and in scenarios ❶, the ref_cnt corresponding to $P3$ would be incremented to 2. Conversely, after computing an ofm element using $P3$ as a source operand, the corresponding ref_cnt value is decremented by 1. When the ref_cnt value reduces to 0, it indicates that the associated Address Mapping Table and OCM resources can be safely released. Otherwise, the related table entries cannot be allocated.

Lifecycle Analysis demonstrates exceptional efficiency in handling fused operators containing residual connections. For example, in Code 2, both `Conv1` and `Ele+` use $A0$ as a source operand. After the Fused function is configured, the ref_cnt of the P.id($P1$) corresponding to $A0$ is set to 2. This mechanism ensures that $A0$ is only released after

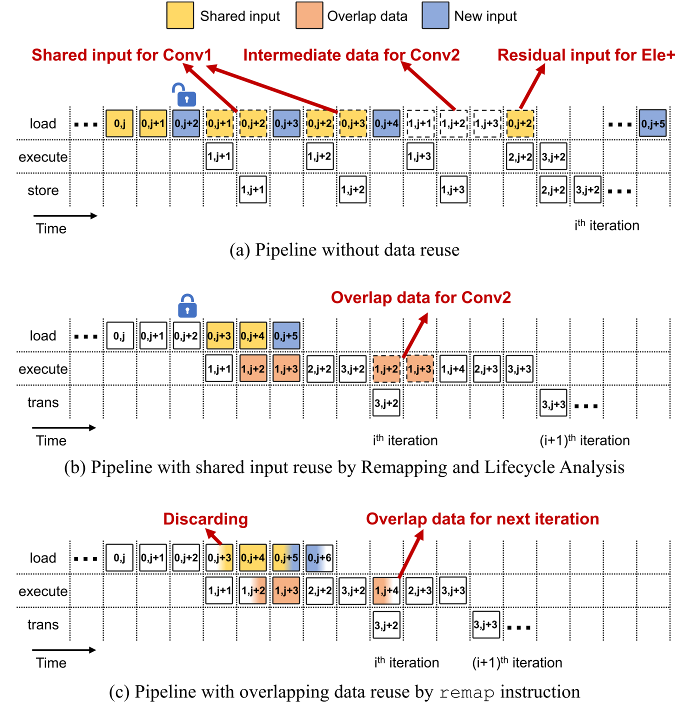


Fig. 6. (a) There are three types of unnecessary stall: reloading shared inputs, reloading overlap data, and accessing long-latency reuse data [e.g., tile at (0, $j+2$)]. (b) Remapping eliminates stalls from reloading shared inputs. Lifecycle Analysis eliminates stalls caused by long-latency reuse data by locking this reuse data until all associated computations are completed. However, stalls due to reloading overlap data between two adjacent iterations persist. (c) `Remap` instruction reuses overlap data in OCM across iterations via redirecting the corresponding A.id to the correct data location. Nonreusable data is discarded immediately, freeing its OCM space for new data.

all associated computations are completed, rather than being freed immediately after its first use. The entire process is performed automatically by hardware, requiring no explicit intervention from the programmer.

C. Pipeline With Remapping

To clearly illustrate the impact of our Remapping and Lifecycle Analysis techniques, Fig. 6 shows the pipeline behavior of the residual block (#0) from Code 2. The pipeline

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

TABLE I
RESOURCE UTILIZATION OF VCU118 ON INT8 ResNet18

	DSP	BRAM	LUT(K)	Power(W)
Remapping	0	0	80	3.66
MACs	4096	0	347	13.73
OCM	0	1088	84	9.15
Others	0	12	67	4.58
Total	4096(60%)	1100(51%)	578(49%)	31.12

consists of three phases: load, execute, and trans (store output to #1). Execute phase concludes `remap`, `Conv1`, `Conv2`, and, `Ele+`, with `remap` not consuming time. For simplicity, all macro instructions are modeled with identical execution times, represented as uniform-width boxes, where numerical labels within boxes correlate to row numbers in Fig. 1.

Fig. 6(b) illustrates that the Remapping resolves the redundant loading of shared input automatically. Additionally, since the shared input is already in the OCM and the false dependency has been addressed, the `load` instruction for the $(i+1)$th iteration can be executed, as soon as the operands are ready. The overlap data generated by `Conv1` in the ith cycle cannot be directly used for the next iteration unless it is specifically saved. To address this issue, fused-layer CNN accelerators [16], [18], [33] provide a byte-level reuse strategy, along with a dedicated buffer to store intermediate data. We propose a `remap` instruction utilizing mRegs for efficient row-level data reuse. As shown in Code 2, after optimizing the Fused function with `remap` instructions, the recomputation problem of overlap activation [dashed orange boxes in Fig. 6(b)] between two adjacent iterations has been covered.

V. EXPERIMENTS

A. Implementation

Our design is implemented in Verilog using Xilinx Vivado 2019.1 and is synthesized at 200 MHz on the VCU118 evaluation platform. To balance resource constraints and computational demands, the accelerator integrates 8 cores. Each core contains a 256 kB coefficient OCM for storing feature maps, an additional 256 kB of BRAM for storing weights, and a further 64 kB for other parameters. We adopt the core design from our previous work [34], where each core contains 2048 MACs (16 × 16 × 8 configuration), mapped to 512 DSP48 slices. By operating DSP48 at double-clock frequency with segmented bit-width processing, each DSP is equivalent to 4 INT8 multiplier–accumulators. Table I details FPGA resource utilization, indicating Remapping occupies 13.8% of LUT resources while accounting for 11.8% of total power consumption. As shown in Fig. 7, Row-wise Tiling reduces logic power by 2.0 W through optimized control circuitry, while Remapping decreases I/O power by 3.4 W via minimized off-chip accesses despite a 0.9-W logic overhead. Lifecycle Analysis further reduces I/O power by 0.36 W. Collectively, these optimizations strategically enhance computational efficiency, increasing DSP power allocation from 20.0% to 29.2% of total power.

In experiments, our FPGA accelerator is compared against other FPGA accelerators, the NVIDIA RTX A6000 GPU, and the Intel i7-10700K CPU. We also examine the impact

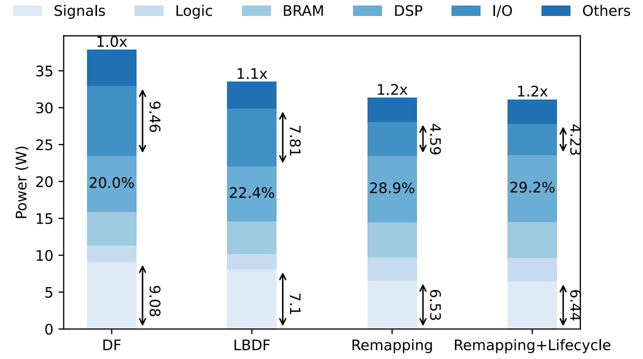


Fig. 7. Total power reduces by 21% through combined I/O power savings (51%) from Row-wise Tiling and Remapping (despite 0.9-W logic overhead), while Row-wise Tiling additionally achieves 22% logic power reduction, collectively improving DSP efficiency.

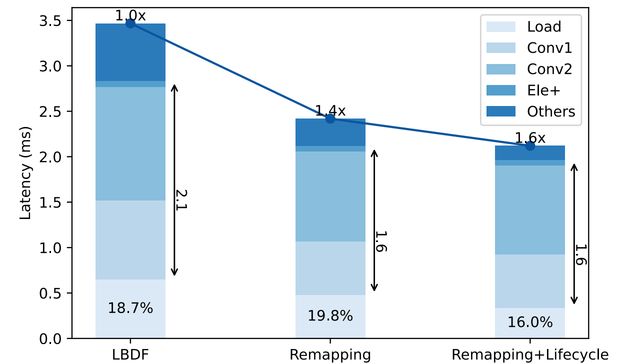


Fig. 8. Execution time of ResNet18 under different approaches. Remapping reduces both memory access latency and compute latency. Lifecycle Analysis further reduces memory access latency, decreasing it from 19.8% to 16.0% of the total latency. “Others” include downsampling layers, linear layers, and reuse pointer generation.

of different optimization strategies—such as Row-wise Tiling, Remapping, and Lifecycle Analysis. Additionally, the experiment employs the roofline model to illustrate AI across platforms and includes a comparison of lines of code to assess the effectiveness of design complexity optimization.

B. Performance of Different Optimization

To discern the performance advantages conferred upon our accelerator by Row-wise Tiling, Remapping, and Lifecycle Analysis individually, we conducted comparative experiments on ResNet18. To overcome the differences in implementation platforms, we re-implemented the 6-level convolution loop (without layer loop and batch loop) of LBDF [18] at the byte level. The comparison approaches include: LBDF, mReg-based Remapping, and Remapping+Lifecycle. As shown in Fig. 8, the Remapping mechanism reduces memory access latency by 0.17 ms and computation overhead by 0.55 ms. Through these optimizations, ResNet18’s load time decreases from 0.65 to 0.34 ms, where 0.14 ms (21.5%) is from residual input reuse.

C. Inference Performance Comparison

Tables II and III systematically compare our accelerator’s performance against prior implementations across multiple neural networks. While previous studies emphasized data

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

TABLE II
COMPARISON BETWEEN THE PROPOSED FRAMEWORK ON MobileNet V2

Platform		CPU	ASIC	FPGA				
		i7-10700K	TECS'21 [35]	FPL'21 [36]	TCASI'21 [37]	VLSI'21 [18]	TCAD'23 [38]	Ours
			SEAD	VC709 (XC7V690t)	ZCU102 (ZU9EG)	VCU118 (XCVU9P)	ZCU102 (ZU9EG)	VCU118 (XCVU9P)
Technology		14nm	90nm	28nm	16nm	16nm	16nm	16nm
Frequency(MHz)		3800	300	150	200	200	333	200
Strategies	Depth First		✔	✔	✔		✔	
	Row-wise					✔		✔
Precision		INT8	INT8	INT8	INT8	INT8	INT8	INT8
DSP Utilization		-	-	2160	576	4096	1283	4096
Power(W)		118.12	2.65	11.35	-	29.56	-	28.92
Throughput (GOPS)		76.9	-	181.8	230.4	1252.1	≈ 1153.1	**1873.9**
DSP Efficiency (GOPS/DSP)		-	-	0.08	0.39	0.31	**0.79**	0.46
Energy Efficiency (GOPS/W)		0.91	-	16.02	-	42.49	-	**65.15**
FPS		177	769	302	382	2072	1910	**3090**
Latency(ms)		5.62	1.30	3.31	≈ 2.62	0.48	≈ 0.52	**0.32**
Off-chip Reduction(%)		0.8	-	-	-	36.9	-	**43.1**

TABLE III
COMPARISON BETWEEN THE PROPOSED FRAMEWORK ON ResNet18/50/152

Platform		GPU	ASIC	FPGA								
		TACO'25 [39]	TECS'21 [35]	HPCA'19 [40]	FPGA'19 [41]	VLSI'21 [18]	TCASI'22 [19]		TCASII'23 [42]	Ours		
		A6000	SEAD	VC707 (XC7VX485t)	VCU118 (XCVU9P)	VCU118 (XCVU9P)	KCU1500 (XCKU040)		VCU118 (XCVU9P)	VCU118 (XCVU9P)		
Technology		4nm	90nm	28nm	16nm	16nm	16nm		16nm	16nm		
Frequency(MHz)		2525	300	150	125	200	200		200	200		
Strategies	Depth First	✔	✔	✔	✔		✔		✔			
	Row-wise					✔				✔		
Precision		FP32	INT8	INT16	INT16	INT8	INT8		FP16	INT8		
DSP Utilization		-	≈ 2048	2800	5489	4096	2240		2320	4096		
Newwork		ResNet50	ResNet50	ResNet152	ResNet50	ResNet18	ResNet50	ResNet152	ResNet18	ResNet18	ResNet50	ResNet152
Power(W)		≈ 100—300	-	21.64	-	33.51	-	-	-	31.12	33.53	33.77
Throughput (GOPS)		1248.6	-	608.3	1235	1555.4	1006	1163	≈ 334.0	**2476.2**	**2913.5**	**2671.0**
Normalized Throughput (MACs/s)		≈ **3800.0** (INT8)	1674.0	1284.0 (INT8)	1872.0 (INT8)	1074.6	655.2	843.2	461.6 (INT8)	**1698.2**	1853.6	**1844.8**
DSP Efficiency (GOPS/DSP)		-	-	0.22	0.23	0.38	0.45	0.52	0.14	**0.60**	**0.71**	**0.65**
Energy Efficiency (GOPS/W)		≈ 4.2—12.5	-	28.1	-	46.4	-	-	-	**79.5**	**86.9**	**79.1**
Latency(ms)		≈ 8.0	4.54	35.2	8.12	3.35	11.7	26.8	15.6	**2.12**	**4.10**	**12.25**
Off-chip Reduction(%)		-	-	47.6	-	40.4	60.6	56.7	-	**72.0**	**65.7**	**68.9**

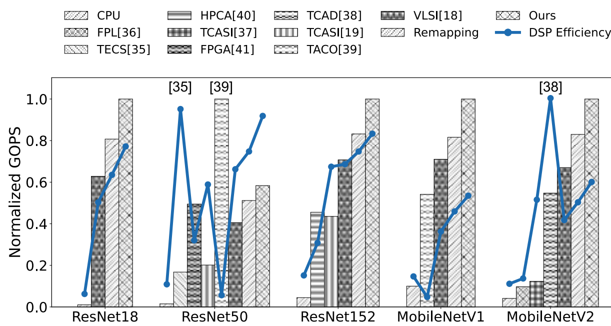


Fig. 9. Our accelerator surpasses most prior works in GOPS and MAC efficiency. While RiSA [35] matches our MAC efficiency via ASIC-specific systolic arrays, a customized MobileNetV2 accelerator [38] achieves superior DSP efficiency at the cost of generality. OptiFX [39] exploits GPU TOPS for ResNet50 throughput but suffers from poor energy efficiency.

locality optimization, few provided measurable metrics for reduced off-chip memory traffic. For standardized evaluation, we employ the Data Movement$_{\text{baseline}}$ benchmark from Shortcut Fusion [19], which isolates layer-wise memory transactions by prohibiting interlayer reuse. As shown in Fig. 9, our design demonstrates superior normalized performance and MAC efficiency compared to previous implementations. Although Jiang et al. [38] achieved higher DSP efficiency for MobileNetV2 through dedicated pipelines and single dequantization, our co-designed architecture enables broader model support and storage-efficient residual connections, rather than copying inputs to deep FIFOs. Similarly, while OptiFX [39] achieves the best ResNet50 throughput using high-clock A6000 hardware (14× our TOPS), it yields substantially lower energy efficiency and MAC efficiency than FPGA-based accelerators performing equivalent workloads.

As shown in Table IV, our Remapping mechanism streamlines data reuse in shifted-window overlaps in the Swin Transformer [24], achieving 2.5× higher energy efficiency compared to Wang et al. [43]. We further implement multi-precision support (BF16, SFP8 [44], and NF4 [45]) validated on MobileViT [46]. While NF4 quantization reduces model storage, its SFP8 dequantization during inference yields limited computational gains due to conversion overhead. At double-clock frequency, each DSP achieves two parallel BF16 multiplications (mantissa-constrained) and four parallel SFP8 multiplications.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
COMPARISON BETWEEN THE PROPOSED FRAMEWORK ON THE SWIN TRANSFORMER AND MobileViT

	TCAD'22 [43]	Ours			
Platform	Alveo U50 (XCVU13P)	VCU118 (XCVU9P)			
Technology	16nm	16nm			
Frequency(MHz)	300	200			
Strategies	Tiling	Row-wise Tiling			
DSP Utilization	2420	4096			
Network	Swin-T	Swin-T	MobileViT		
Precision	FP16	BF16	BF16	SFP8	NF4
Throughput (GOPS)	309.6	701.4	994.5	1382.4	1628.2
DSP Efficiency (GOPS/DSP)	0.13	0.17	0.24	0.34	0.39
Energy Efficiency (GOPS/W)	7.94	19.9	29.9	43.5	48.8
Off-chip Reduction(%)	-	33.3	31.8	32.4	32.0

TABLE V
COMPARISON BETWEEN THE PROPOSED FRAMEWORK ON THE GCN

Platform	FPGA	
	TPDS'23 [47]	Ours
	Alveo U250 (XCU250)	VCU118 (XCVU9P)
Technology	16nm	16nm
Frequency(MHz)	300	200
Precision	FP32	INT8
BRAM Utilization	1853 (69%)	1100 (51%)
LUT Utilization	778K (45%)	578K (49%)
DSP Utilization	10240 (83%)	4096 (60%)
Throughput (GOPS)	469.5 (INT8)	87.4
DSP Efficiency (GOPS/DSP)	0.05	0.02
Energy Efficiency (GOPS/W)	-	3.4
Latency(ms)	2.13	2.86
Off-chip Reduction(%)	-	8.1

We also compare our accelerator against GraphAGILE [47], a GNN-oriented accelerator that employs edge-centric processing and shuffle networks to accelerate sparse matrix operations. As shown in Table V, GraphAGILE achieves approximately 2.14× higher DSP efficiency than our design when executing the GCN [48] on the Cora dataset. This gap reflects a fundamental design tradeoff—our accelerator intentionally omits the specialized circuitry (e.g., butterfly networks) required for sparse computation to maximize computational density and energy efficiency for dense workloads such as CNNs and ViTs.

D. Roofline Model Analysis

We have unified i7-10700K and A6000 [39] into the roofline model to provide a more intuitive comparison of the performance differences across different platforms in network inference. The theoretical INT8 performance of the i7-10700K is 1.3 TOPS with a bandwidth of 45.8 GB/s. The A6000, on the other hand, has a theoretical FP32 performance of 91.1 TOPS and a bandwidth of 960 GB/s. Our VCU118-based accelerator achieves a theoretical INT8 fixed-point performance of 6.5 TOPS and a bandwidth of 38.4 GB/s. For comparison, we selected MobileNetV2 and ResNet18/50/152 models. On the i7-10700K, we used the INT8 models provided by PyTorch.

As shown in Fig. 10, while our memory bandwidth is relatively lower, causing a rightward shift of the ridge point in the roofline model, our approach—combining Row-wise Tiling and Remapping—enables exceptionally high AI. Our accelerator achieves the highest AI with ResNet18, as it employs solely 3 × 3 kernels in the residual block, which maximizes data reuse opportunities. In contrast, ResNet50 and ResNet152 employ 1 × 1 kernels in both the input and output layers, diminishing their overall AI. ResNet152 partially compensates for this through its 75% operators with 1024–2048 output channels (versus 54% in ResNet50). However, ResNet variants fundamentally outperform MobileNetV2, where the prevalent use of 1 × 1 kernels combined with a maximum of 320 output channels.

i7-10700K achieves higher AI on lightweight MobileNetV1 than the A6000, leveraging its multilevel cache hierarchy and latency-oriented prefetch mechanism. While OptiFX [39] enhances ResNet50 performance via operator fusion, the optimization gain is limited on MobileNet due to poor resource utilization. Analyzing AI and performance trends on the i7-10070K proves more challenging due to its intricate cache hierarchy and hybrid data reuse strategies, which introduce nonlinear interactions between model characteristics and hardware utilization. Our accelerator not only delivers significant performance gains but also brings the performance of different models closer to the theoretical peak computing power of the architecture than others, demonstrating higher computational efficiency.

Table VI compares the actual AI (AI_{act}) across different platforms. As formulas (1)–(4) reveal, AI_{act} mainly depends on the locality optimization of feature maps rather than weights. R_w relies on the input shape (H and W), so models with the same block structures (e.g., ResNet variants) have similar R_w values. Weights reside in OCM during inference, reducing their opportunity to further enhance AI_{act}. Thus, for the same OCM management strategies, a higher R_f offers greater potential to boost AI_{act}. Our accelerator shows superior AI_{act}/AI_{max} ratios, especially in ResNet18, which has the highest R_f. Although MobileNetV1's exclusive 3 × 3 kernels yield a higher AI_{max} than MobileNetV2, the lack of inverted bottlenecks reduces activation reuse efficiency, leading to lower AI_{act}/AI_{max} across different platforms.

E. Fusion Configuration Analysis

Similar to the DF approach [16], our design prioritizes deeper layer fusion over larger batch sizes to maximize on-chip data reuse. To assess the potential impact of OCM or Free List pressure, we evaluate ResNet18 under varying batch sizes and fusion depth. As shown in Table VII, enabling preloading via Remapping and Lifecycle Analysis yields nearly identical memory utilization, energy efficiency, and DSP efficiency across different batch sizes. Larger batches are decomposed into sequential single-batch executions with pipelined preloading, which ensures that input data becomes available for computation at the earliest possible moment. In contrast, shallow fusion configurations such as Fusion Depth = 4 underutilize the DF advantage and therefore achieve lower performance relative to deeper fusions. Overall, potential memory pressure is mitigated by two mechanisms: 1) pipelined preloading, which overlaps data movement with

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

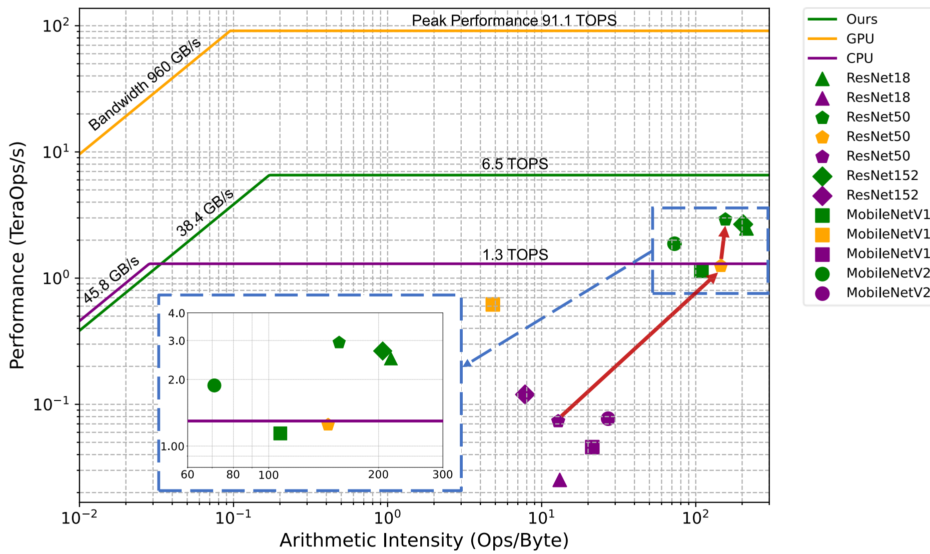


Fig. 10. Comparison between RTX A6000, i7-10700K, and our accelerator in the roofline model. Purple is for i-7 10700K, yellow is for RTX A6000, and green is for ours. Within the lowest bandwidth, our accelerator achieves the highest AI and the best performance across different models. This is attributed to improved data locality and efficient OCM management, which minimizes computational resources idleness. On our accelerator, ResNet18 achieves optimal AI due to higher data reuse enabled by its 3 × 3 kernels. In contrast, ResNet50 and ResNet152, which rely heavily 1 × 1 kernels, experience a sharp reduction in data reuse. Due to deeper output channels, ResNet152 achieves the second-highest AI. MobileNetV2, limited by its 1 × 1 kernels and a maximum of 320 output channels, exhibits significantly lower AI compared to ResNet models.

TABLE VI
COMPARISON OF ACTUAL VERSUS THEORETICAL AI

Network	Average R_{f}	Average R_{w}	$AI_{\max}$	$AI_{\mathrm{act}}/AI_{\max}(\%)$					
				i7-10070K	A6000 [39]	VCU118 [18]	VC707 [40]	KCU1500 [37]	VCU118 (Ours)
ResNet18	882.8	190.6	303.8	7.9	-	32.6	-	-	**69.4**
ResNet50	440.8	191.3	295.1	4.3	49.3	-	-	41.8	**50.7**
ResNet152	558.3	208.5	374.5	6.1	-	-	32.2	38.9	**54.2**
MobileNetV1	27.8	129.1	261.6	15.9	1.8	-	-	-	**39.6**
MobileNetV2	66.4	168.1	169.0	27.8	-	23.8	-	-	**42.0**

TABLE VII
OCM UTILIZATION AND HARDWARE EFFICIENCY OF ResNet18 UNDER DIFFERENT BATCH SIZES AND FUSION DEPTHS

Preload-ing	Batch Size/Core	Fusion Depth (Blocks/Core)	Memory Utilization (%)	Energy Efficiency (GOPS/W)	DSP Efficiency (GOPS/DSP)
✘	1	8	81.1	72.2	0.53
✔	**1**	8	93.6	79.5	0.60
✔	**8**	**8**	93.6	79.4	0.60
✔	**16**	8	93.6	79.5	0.60
✔	8	**16**	93.6	79.5	0.60
✔	8	**4**	69.8	61.2	0.44

TABLE VIII
OCM UTILIZATION AND FUSION DEPTH FOR DIFFERENT MODELS ON THE IMAGENET DATASET (INPUT SIZE: 256 × 256)

Network	Batch Size/Core	Fusion Depth (Blocks/Core)	Residual Block Size (KB)	Memory Utilization (%)
ResNet18	1	8	28–32	81.1
ResNet50	1	4	52–56	86.7
ResNet152	1	4	52–56	87.3
MobileNetV1	2	5–8	12–48	85.9
MobileNetV2	2	3–4	36–92	84.9

computation to hide memory latency and 2) compile-time OCM safety enforcement, which bounds the total fused layer size within the OCM limit. For instance, a nominal Fusion Depth of 16 exceeds this constraint and is automatically partitioned into OCM-safe segments.

Effective utilization of the 256 kB feature map OCM requires careful balancing between layer fusion depth and OCM allocation. As quantified in Table VIII for IMAGENET (256 × 256 input), ResNet18's residual blocks (28–32 kB/block) enable full model fusion (8 blocks) under single-core batch size 1, requiring only input/output off-chip access during inference. Fig. 11 demonstrates that the 4 kB base unit within a 256 kB OCM configuration provides a tradeoff across different DNN architectures.

F. Platform Adaptability Analysis

To evaluate architectural adaptability for ASIC platforms, we implement our design in TSMC 28-nm technology at 800 MHz using Synopsys Design Compiler and PrimeTime PX. As shown in Fig. 12, Remapping occupies under 3% of both area and power resources. The ASIC implementation achieves similar off-chip data reduction to the FPGA version while operating at significantly higher clock frequencies. Although GPGPUs offer higher peak throughput, Table VI reveals their energy inefficiency from suboptimal operand scheduling. While Remapping could improve AI, its GPU implementation would require fundamental memory hierarchy reconstruction—despite these architectural barriers, the

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

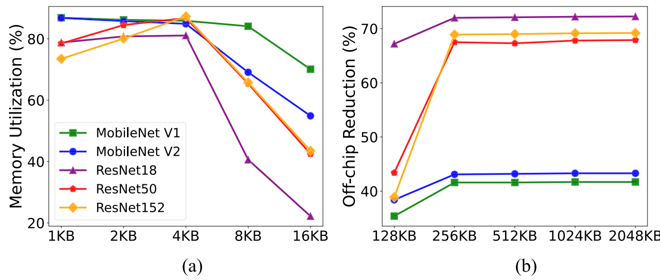


Fig. 11. (a) 4 kB base unit design benefits most models, as shown by memory utilization comparisons across different mReg base unit sizes. (b) Increasing the OCM size to 256 kB achieves optimal configuration, as further expansion does not significantly enhance off-chip reduction.

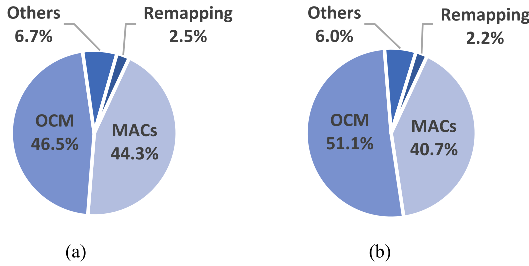


Fig. 12. (a) Area and (b) power breakdown of the 28-nm ASIC implementation at 800 MHz.

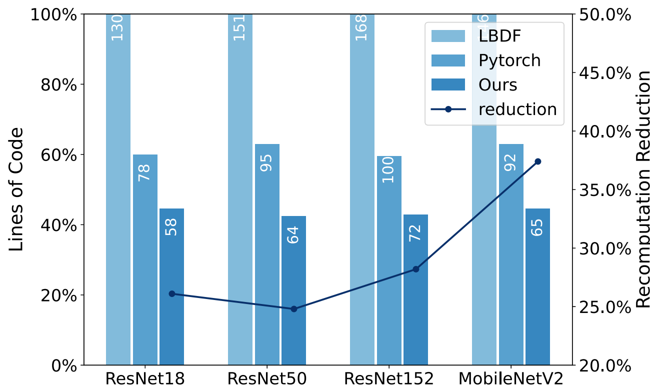


Fig. 13. mISA enables programmers to perform complex operations with fewer instructions, significantly reducing lines of code. MobileNetV2 achieves greater recomputation reduction than ResNet variants on our accelerator, due to its inverted bottleneck enhances the benefits of overlap reuse through remapping.

potential performance-per-watt improvements warrant further investigation.

G. Programming Complexity on Different Format

As described in Section III-B, our kernel functions are constructed using `regs`, `args`, and `launch` instructions. This methodology yields a compact implementation compared to traditional operator-based functions for a 6-D computational kernel, or PyTorch module-based operators. This efficiency is achieved by leveraging row tile-based mRegs as a basic operand in mISA, as the Fused function depicted in Code 2. Unlike conventional byte granularity encoding schemes in [16] and [18], mISA uses a single `remap` instruction to facilitate row-level overlap data reuse, eliminating the need for fine-grained address calculations. As quantified in Fig. 13, deeper layer fusion, just like ResNet18, minimally impacts code complexity, because repetitive intermediate residual blocks are modularized as reusable functions rather than flattened instructions sequentially. Notably, MobileNetV2 achieves greater recomputation reduction than ResNet variants on our accelerator due to its inverted bottleneck design, where intermediate layers expand to 6× the input channel count, amplifying the benefits of overlap reuse via Remapping.

VI. Related Work

Tiling and data reuse techniques have dramatically mitigated off-chip access. Techniques like SmartShuttle's dynamic tiling [30], Shortcut Mining's decoupled physical-logic buffer [40], and Shortcut Fusion's Reuse-Aware Static Memory Allocation [19] significantly reduce off-chip access. LBDF [18] employs Row-wise Tiling to simplify overlap data reuse. However, these approaches fundamentally lack depthwise convolution support due to channel-parallel conflicts. Comparatively, Li et al. [37] supported both convolution types through reconfigurable processing elements with partial sum memory overhead, while Jiang et al. [38] employed dedicated MobileNetV2 pipelines, sacrificing architectural flexibility. Our key innovation, the Row-wise Tiling-based Remapping mechanism, supports automatic data reuse without physical data movement.

Processing-in-memory (PIM) integrates computation directly within memory arrays, eliminating the energy overhead of data movement [49]. ReRAM-based PIM accelerators offer significant potential for energy-efficient matrix processing [50], as their analog crossbar structures are particularly suitable for storing weights and performing computations for CNN inference. MemUnison [51] utilizes ReRAM to store weights and execute multiply–accumulate operations, while employing Racetrack Memory [52] as a streaming buffer to minimize intermediate data transfers and mitigate ReRAM's write amplification energy penalty. Our design, combined ReRAM-Racetrack approach, can further reduce the computational complexity of convolutional layers.

Stochastic computing (SC) has emerged as a promising approximate computing technique, offering significant reductions in the power consumption for DNNs [53]. By encoding numerical values as probabilistic bitstreams, SC enables ultralow-power arithmetic operations using simple logic gates. Current SC research focuses on mitigating its inherent precision limitations, particularly the severe noise imbalance where small products suffer disproportionately high errors compared to large ones [54]. HSCM [55] addresses this by heuristically partitioning operations into three specialized domains: near-zero products are set to zero, mid-range values use compressed lookup tables for near-binary precision, and large products retain LD-SC's efficiency. This tri-mode approach establishes a new Pareto frontier in energy-accuracy tradeoffs for edge DNN through minimal circuitry. SC presents a compelling research direction for further optimizing computational resources and reducing data access in our work.

VII. Conclusion

To better support Row-wise Tiling, improve OCM utilization, and reduce memory accesses, as well as accommodate the residual connections, we propose a flexible mISA within a hierarchical-ISA combined with a hardware

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

Remapping scheme. This scheme provides a promising approach for efficiently deploying DNNs across diverse DSAs. By systematically exploiting data locality and optimizing resource utilization, the proposed accelerator design significantly improves neural network inference performance. At present, we have established a prototype on an FPGA. In the future, we will further expand the operator library to meet the requirements of more models and explore the adaptability for general-purpose GPU architectures.

REFERENCES

[1] J. Oruh, S. Viriri, and A. Adegun, "Long short-term memory recurrent neural network for automatic speech recognition," *IEEE Access*, vol. 10, pp. 30069–30079, 2022.

[2] A. S. Dhanjal and W. Singh, "A comprehensive survey on automatic speech recognition using neural networks," *Multimedia Tools Appl.*, vol. 83, no. 8, pp. 23367–23412, Aug. 2023.

[3] D. J. Miller, Z. Xiang, and G. Kesidis, "Adversarial learning targeting deep neural network classification: A comprehensive review of defenses against attacks," *Proc. IEEE*, vol. 108, no. 3, pp. 402–433, Mar. 2020.

[4] A. Azab, M. Khasawneh, S. Alrabaee, K.-K.-R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digit. Commun. Netw.*, vol. 10, no. 3, pp. 676–692, Jun. 2024.

[5] T. Y. Lin, P. Dollàr, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 2117–2125.

[6] K. He, D. D. Kim, and M. R. Asghar, "Adversarial machine learning for network intrusion detection systems: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 538–566, 1st Quart., 2023.

[7] W. H. Wen-Mei, D. B. Kirk, and I. El Hajj, *Programming Massively Parallel Processors: A Hands-on Approach*. San Mateo, CA, USA: Morgan Kaufmann, 2022.

[8] L. Xuan, K.-F. Un, C.-S. Lam, and R. P. Martins, "An FPGA-based energy-efficient reconfigurable depthwise separable convolution accelerator for image recognition," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 10, pp. 4003–4007, Oct. 2022.

[9] T. Chen et al., "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 269–284, Apr. 2014.

[10] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[11] Y.-H. Chen, T.-J. Yang, J. S. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[12] C. Hong, A. Sukumaran-Rajam, I. Nisa, K. Singh, and P. Sadayappan, "Adaptive sparse tiling for sparse matrix multiplication," in *Proc. 24th Symp. Princ. Pract. Parallel Program.*, Feb. 2019, pp. 300–314.

[13] G. A. Reina, R. Panchumarthy, S. P. Thakur, A. Bastidas, and S. Bakas, "Systematic evaluation of image tiling adverse effects on deep learning semantic segmentation," *Frontiers Neurosci.*, vol. 14, p. 65, Feb. 2020.

[14] H. Je, D. T. Nguyen, K. Lee, and H.-J. Lee, "An adaptive row-based weight reuse scheme for FPGA implementation of convolutional neural networks," in *Proc. 36th Int. Tech. Conf. Circuits/Systems, Comput. Commun. (ITC-CSCC)*, Jun. 2021, pp. 1–4.

[15] C.-C. Kao, "Optimizing FPGA-based convolutional neural network performance," *J. Circuits, Syst. Comput.*, vol. 32, no. 15, Oct. 2023, Art. no. 2350254.

[16] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused-layer CNN accelerators," in *Proc. 49th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[17] W. Niu, J. Guan, Y. Wang, G. Agrawal, and B. Ren, "DNNFusion: Accelerating deep neural networks execution with advanced operator fusion," in *Proc. 42nd ACM SIGPLAN Int. Conf. Program. Lang. Design Implement.*, Jun. 2021, pp. 883–898.

[18] S. Colleman and M. Verhelst, "High-utilization, high-flexibility depth-first CNN coprocessor for image pixel processing on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 3, pp. 461–471, Mar. 2021.

[19] D. T. Nguyen, H. Je, T. N. Nguyen, S. Ryu, K. Lee, and H.-J. Lee, "ShortcutFusion: From tensorflow to FPGA-based accelerator with a reuse-aware memory allocation for shortcut data," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 6, pp. 2477–2489, Jun. 2022.

[20] E. Valpreda et al., "HW-Flow-fusion: Inter-layer scheduling for convolutional neural network accelerators with dataflow architectures," *Electronics*, vol. 11, no. 18, p. 2933, Sep. 2022.

[21] M. Shi, P. Houshmand, L. Mei, and M. Verhelst, "Hardware-efficient residual neural network execution in line-buffer depth-first processing," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 11, no. 4, pp. 690–700, Dec. 2021.

[22] P. Li, C. Che, and R. Hou, "Nacc-guard: A lightweight DNN accelerator architecture for secure deep learning," *J. Supercomput.*, vol. 80, no. 5, pp. 5815–5831, Mar. 2024.

[23] N. P. Jouppi et al., "Ten lessons from three generations shaped Google's TPUv4i : Industrial product," in *Proc. ACM/IEEE 48th Annu. Int. Symp. Comput. Archit. (ISCA)*. Industrial, Jun. 2021, pp. 1–14.

[24] Z. Liu et al., "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 9992–10002.

[25] J. Wang, Y. Wu, and D. Wang, "SC-GNN: A communication-efficient semantic compression for distributed training of GNNs," in *Proc. 61st ACM/IEEE Design Autom. Conf.*, Jun. 2024, pp. 1–6.

[26] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of DNN dataflow: A data-centric approach," in *Proc. 52nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, Oct. 2019, pp. 754–768.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[28] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[29] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[30] J. Li et al., "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 343–348.

[31] T. Chen et al., "TVM: An automated end-to-end optimizing compiler for deep learning," 2018, *arXiv:1802.04799*.

[32] S. Kim and S.-M. Moon, "Rotating register allocation for enhanced pipeline scheduling," in *Proc. 16th Int. Conf. Parallel Archit. Compilation Techn. (PACT)*, Sep. 2007, pp. 60–72.

[33] K. Goetschalckx and M. Verhelst, "Breaking high-resolution CNN bandwidth barriers with enhanced depth-first execution," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 9, no. 2, pp. 323–331, Jun. 2019.

[34] B. Zhao et al., "REMAP: A spatiotemporal CNN accelerator optimization methodology and toolkit thereof," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1691–1704, May 2023.

[35] H. Cho, "RiSA: A reinforced systolic array for depthwise convolutions and embedded tensor reshaping," *ACM Trans. Embedded Comput. Syst.*, vol. 20, no. 5s, pp. 1–20, Oct. 2021.

[36] S. Yan et al., "An FPGA-based MobileNet accelerator considering network structure characteristics," in *Proc. 31st Int. Conf. Field-Programmable Log. Appl. (FPL)*, Aug. 2021, pp. 17–23.

[37] B. Li et al., "Dynamic dataflow scheduling and computation mapping techniques for efficient depthwise separable convolution acceleration," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 8, pp. 3279–3292, Aug. 2021.

[38] W. Jiang, H. Yu, and Y. Ha, "A high-throughput full-dataflow MobileNetv2 accelerator on edge FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 5, pp. 1532–1545, May 2023.

[39] X. Wang et al., "OptiFX: Automatic optimization for convolutional neural networks with aggressive operator fusion on GPUs," *ACM Trans. Archit. Code Optim.*, vol. 22, no. 2, pp. 1–27, Jun. 2025.

[40] A. Azizimazreah and L. Chen, "Shortcut mining: Exploiting cross-layer shortcut reuse in DCNN accelerators," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 94–105.

[41] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2019, pp. 73–82.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.

[42] R. T. Syed, M. Andjelkovic, M. Ulbricht, and M. Krstic, "Towards reconfigurable CNN accelerator for FPGA implementation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 3, pp. 1249–1253, Mar. 2023.
[43] T. Wang et al., "ViA: A novel vision-transformer accelerator based on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 4088–4099, Nov. 2022.
[44] W. Zhao, Q. Dang, T. Xia, J. Zhang, N. Zheng, and P. Ren, "Optimizing FPGA-based DNN accelerator with shared exponential floating-point format," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 11, pp. 4478–4491, Nov. 2023.
[45] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2023, pp. 10088–10115.
[46] S. Mehta and M. Rastegari, "MobileViT: Light-weight, general-purpose, and mobile-friendly vision transformer," 2021, *arXiv:2110.02178*.
[47] B. Zhang, H. Zeng, and V. K. Prasanna, "GraphAGILE: An FPGA-based overlay accelerator for low-latency GNN inference," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 9, pp. 2580–2597, Sep. 2023.
[48] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.
[49] K. Asifuzzaman, N. R. Miniskar, A. R. Young, F. Liu, and J. S. Vetter, "A survey on processing-in-memory techniques: Advances and challenges," *Memories - Mater., Devices, Circuits Syst.*, vol. 4, Jul. 2023, Art. no. 100022.
[50] X. Yang, B. Yan, H. Li, and Y. Chen, "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.
[51] J. Wang, J. Liu, D. Wang, S. Zhang, and X. Fan, "MemUnison: A Racetrack-ReRAM-combined pipeline architecture for energy-efficient in-memory CNNs," *IEEE Trans. Comput.*, vol. 71, no. 12, pp. 3281–3294, Dec. 2022.
[52] J. Wang, J. Liu, D. Wang, J. An, and X. Fan, "An automatic-addressing architecture with fully serialized access in racetrack memory for energy-efficient CNNs," *IEEE Trans. Comput.*, vol. 71, no. 1, pp. 235–250, Jan. 2022.
[53] H. Sim and J. Lee, "A new stochastic computing multiplier with application to deep convolutional neural networks," in *Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.
[54] S. Asadi, M. H. Najafi, and M. Imani, "CORLD: In-stream correlation manipulation for low-discrepancy stochastic computing," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.
[55] J. Wang, H. Chen, D. Wang, K. Mei, S. Zhang, and X. Fan, "A noise-driven heterogeneous stochastic computing multiplier for heuristic precision improvement in energy-efficient DNNs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 2, pp. 630–643, Feb. 2023.

Zhiwang Huo received the bachelor's degree from Shenzhen University, Shenzhen, China, in 2017, and the master's degree from Xi'an University of Technology, Xi'an, China, in 2021. He is currently pursuing the Ph.D. degree with the College of Artificial Intelligence, Xi'an Jiaotong University, Xi'an.

His current research interests include deep neural network (DNN)-based accelerator computer architecture.

Wenzhe Zhao (Member, IEEE) received the bachelor's, master's, and Ph.D. degrees from Xi'an Jiaotong University, Xi'an, China, in 2005, 2008, and 2014, respectively.

From 2011 to 2013, he was a Visiting Student with Rensselaer Polytechnic Institute, Troy, NY, USA. He is currently an Assistant Professor with the College of Artificial Intelligence, Xi'an Jiaotong University. His current research interests include computer architecture, deep neural network (DNN)-based computing architecture, and VLSI design.

Qiwei Dang received the bachelor's degree in information engineering from Xi'an Jiaotong University, Xi'an, China, in 2020, where he is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence and Robotics, Department of Artificial Intelligence.

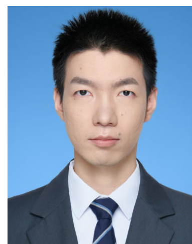
His research interests include computing architecture and domain-specific accelerator design.

Chengyu Ma received the bachelor's degree from Northwestern Polytechnical University, Xi'an, China, in 2020. He is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an.

His current research interests include deep neural network (DNN)-based accelerators and model compression.

Guoming Yang received the bachelor's degree from Northeastern University, Shenyang, China, in 2019, and the master's degree from Xi'an Jiaotong University, Xi'an, China, in 2022, where he is currently pursuing the Ph.D. degree with the Institute of Artificial Intelligence and Robotics, Department of Artificial Intelligence.

His current research interests include deep neural network (DNN)-based computing architecture and VLSI design.



Gelin Fu (Member, IEEE) received the bachelor's degree in control science and engineering from Chongqing University, Chongqing, China, in 2019. He is currently pursuing the Ph.D. degree with the College of Artificial Intelligence, Xi'an Jiaotong University, Xian, China.

His research interests include system modeling and optimization and computer architecture.

Tian Xia (Member, IEEE) received the Ph.D. degree from the National Institute of Applied Sciences, Bellignat, France, in 2016.

He has been an Assistant Professor with the College of Artificial Intelligence, Xi'an Jiaotong University, Xi'an, China, since 2019. His current research interests include cloud-computing virtualization and innovative parallel computation architecture.

Pengju Ren (Member, IEEE) received the bachelor's and Ph.D. degrees from Xi'an Jiaotong University, Xi'an, China, in 2004 and 2012, respectively.

From 2009 to 2011, he was a Visiting Ph.D. Student with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA. He is currently a Professor with the College of Artificial Intelligence, Xi'an Jiaotong University. His current research interests include on-chip networks and computer architecture.

Authorized licensed use limited to: Xian Jiaotong University. Downloaded on June 23,2026 at 12:24:07 UTC from IEEE Xplore. Restrictions apply.