JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL. 23, NO. 5, OCTOBER, 2023
https://doi.org/10.5573/JSTS.2023.23.5.322
ISSN(Print) 1598-1657
ISSN(Online) 2233-4866

In-depth Survey of Processing-in-memory Architectures for Deep Neural Networks

Ji-Hoon Jang*, Jin Shin*, Jun-Tae Park, In-Seong Hwang, and Hyun Kim

Abstract—**Processing-in-Memory (PIM) is an emerging computing architecture that has gained significant attention in recent times. It aims to maximize data movement efficiency by moving away from the traditional von Neumann architecture. PIM is particularly well-suited for handling deep neural networks (DNNs) that require significant data movement between the processing unit and the memory device. As a result, there has been substantial research in this area. To optimally handle DNNs with diverse structures and inductive biases, such as convolutional neural networks, graph convolutional networks, recurrent neural networks, and transformers, within a PIM architecture, careful consideration should be given to how data mapping and data flow are processed in PIM. This paper aims to provide insight into these aspects by analyzing the characteristics of various DNNs and providing detailed explanations of how they have been implemented with PIM architectures using commercially available memory technologies like DRAM and next-generation memory technologies like ReRAM.**

Index Terms—**Processing-in-memory, deep learning, next-generation memory, near-memory computing, deep neural network**

Manuscript received Jun. 27, 2023; reviewed Aug. 22, 2023; accepted Sep. 10, 2023
Research Center for Electrical and Information Technology, Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul 01811, Korea
E-mail : hyunkim@seoultech.ac.kr (Corresponding author: Hyun Kim)
Ji-Hoon Jang and Jin Shin contributed equally to this work.

I. INTRODUCTION

With the emergence of the backpropagation algorithm [8] and multilayer perceptron [3], deep neural networks (DNNs) have demonstrated outstanding performance in various fields [1, 2, 110]. However, they face the challenge of exponentially increasing computational load as the number of learnable parameters grows. This poses a significant obstacle to the practical implementation of DNN models in terms of processing speed and power consumption [113]. To tackle these issues, parallel processing devices such as graphics processing units (GPUs) and neural processing units (NPUs) [4] are being utilized, and researchers are actively exploring optimized acceleration algorithms for each device [5]. However, modern computer architectures based on the von Neumann architecture still have limitations regarding DNN processing. Specifically, a substantial portion of the power consumption, up to 75%, is attributed to loading parameters for DNN operations (*e.g.,* feature maps and weights) from external memory, such as dynamic random-access memory (DRAM), to the processor, or storing them back to memory [9, 10, 115]. To address this issue, processor-in-memory (PIM) architecture has emerged as a promising technology [6]. By integrating computing and memory units at the processing element (PE) level, PIM significantly reduces latency associated with data transmission and enhances data processing efficiency [112]. This integrated architecture has the potential to significantly reduce energy consumption during memory access, thereby enhancing the efficiency of applications that require high-performance computing [7].

This survey explores diverse PIM architectures and

methodologies for enhancing PIM performance in different memory types. It analyzes the characteristics of various DNN models, including convolutional neural networks (CNNs), graph neural networks (GNNs), recurrent neural networks (RNNs), and transformer models. The focus is on optimizing data mapping and dataflows within the context of PIM, providing valuable insights into efficient handling of DNNs. This comprehensive study aims to deepen researchers' understanding of the connection between DNNs and PIM, opening up new avenues for future AI research and advancements.

Section II shows the background of this work. Section III presents the PIM architectures for DNNs, and Section IV concludes this paper.

II. BACKGROUND

1. Processing in Memory (PIM)

In this section, we categorize PIM into two main types: processing-near-memory (PnM) and processing-using-memory (PuM), based on where the data processing takes place, as classified in [116]. We then provide an overview of the characteristics associated with each category. Section III highlights representative studies that fall within these respective categories, offering valuable insights into the field.

1) Processing-using-Memory (PuM)

PuM refers to a technology that performs data operation processing within the memory itself. Previous research on in-memory computing architectures [81-99] has paved the way for PuM, which leverages memory cell structures or modifies the minimum unit memory structures to carry out computational functions directly in the memory. This approach helps alleviate the bottleneck between the processor and memory, resulting in minimal overhead and without requiring additional space. However, due to the need for compatibility with existing memory structures, PuM has limitations in terms of the range of operations it can support. Typically, PuM implementations focus on bitwise operations like AND, OR, XOR, and NOR [81, 82, 84, 85]. Implementing PuM technology is a notable challenge, particularly with DRAM, as it is already highly advanced and optimized for its read/write operations. DRAM-based PuM designs typically incorporate simple logic after sense-amplifier (SA) to perform small-scale operations [81, 82, 84, 85]. On the other hand, static random-access memory (SRAM)-based studies have explored analog operator units, taking advantage of relatively fewer spatial constraints, which enable support for more complex operations [86-89]. Another promising candidate for PuM is resistive random-access memory (ReRAM), a next-generation memory that efficiently carries out multiple-accumulate (MAC) operations by leveraging the crossbar structure [84, 90-94, 100].

2) Processing-near-Memory (PnM)

PnM is a technology that performs data operation processing using dedicated operation logic located close to the memory. Research is being conducted to apply PnM to various memory types [48-80, 118, 119]. The goal of PnM is to enhance system performance by minimizing data movement between the processor and memory for computation or by optimizing the data frame structure. Unlike PuM, PnM allows for the implementation of more complex operations, resulting in a wider range of methods. However, it should be noted that the improvement in memory bottleneck is not as significant as with PuM, as it still involves more data movement between the processors and memory. Additionally, PnM has the drawback of requiring a higher area overhead. Many studies focusing on SRAM-based PnM have explored implementing multiple operators in parallel near an SRAM cell array to enable parallel operation [48, 49, 89]. On the other hand, DRAM-based studies have primarily focused on achieving parallel operations by efficiently utilizing bank parallelism [50-59].

2. Various Deep Neural Networks

Various types of DNNs have been proposed based on diverse inductive biases (*e.g.,* locality, sequentiality, arbitrariness) [6]. The computational process of DNNs can be broadly classified into two categories: linear operations (*i.e.,* operations such as matrix-vector multiplication (VMM) and matrix-matrix multiplication (MatMul) that utilize MAC as the fundamental unit of operation) and non-linear operations (*i.e.,* ReLU [11],

tanh [12], and sigmoid [13]). Linear operations are used to calculate the weights of the input features and hidden layers for feature extraction. Generally, strategies to enhance the performance of DNNs involve stacking deeper layers [14, 15] or widening a layer [16]. Owing to these strategies, MAC operations constitute the largest share of all operations in DNNs, leading to potential issues. 1) Its computationally intensive nature can lead to high latency, necessitating high parallelism. 2) Frequent memory access and data movement for the weights and input data can result in substantial power consumption, creating bottlenecks. 3) Substantial hardware design costs can occur in an effort to mitigate these issues. In this subsection, DNNs are classified based on their network structure and inductive bias.

1) Convolutional Neural Network (CNN)

A CNN is a type of DNN specialized for processing two-dimensional (2D) image data. It typically consists of convolution (CONV) layers, normalization layers, pooling layers, and activation functions. CNN is capable of extracting and learning features from the input that are then used to perform specific tasks. In particular, within the CONV layer, a multichannel kernel extracts local features from an image by sliding across the input at a certain stride interval [17]. The CONV operation constitutes a significant part of the CNN model and requires a high external memory bandwidth owing to the need to load multichannel intermediary features and a large number of weights [18, 19, 111]. Traditionally, CNNs have been used as the backbone for vision tasks. Recently, they have been used as embedding layers for token extraction in transformer-based networks [20].

2) Graph Neural Network (GNN)

A GNN is a general framework for operating on complex structures represented by graphs, consisting of nodes (vertices) and the edges connecting them. Unlike fixed formats such as 2D or 3-dimensional (3D) data, it mainly deals with the more abstract relationships used in workloads, such as social networks or media influence. Traditional algorithms that use graph data (*e.g.,* search [21], shortest path finding [22], spanning tree [23], and clustering [24]) cannot inherently evolve into research on the graph structure itself because they require prior knowledge of the input graph. However, GNN applies artificial neural networks directly to the graph data, enabling predictions at the node, link, and graph levels. Through iterative parameter updates, learnable parameters are trained to better represent the relationships between adjacent nodes within a graph [25]. Additionally, a GNN is capable of mimicking most deep learning models (*e.g.,* CNN, RNN, and self-attention mechanisms) by applying the relationships between nodes. In particular, the graph convolutional network (GCN), a prominent network derived from the GNN, focuses on aggregating information from neighboring nodes, similar to how a 2D CNN learns the locality between pixels, as shown in Fig. 1, which can reduce the computational complexity of GNNs with little performance degradation as they do not include unnecessary node connections [26, 27].

3) Recurrent Neural Network (RNN)

An RNN is designed for tasks that involve learning patterns in sequential data or understanding contextual information in sentences. They are designed to retain previous information and incorporate new information, thereby effectively learning the patterns or dependencies in sequential data. RNN architecture employs recurrent neurons that sequentially process each element of the data sequence. The result of the operation in the previous hidden state is used as an input for the calculations in the current hidden state. This allows for the simultaneous processing of the output of the previous hidden state and the current input data, enabling the network to understand the interrelationships and dependencies within the data.

Long short-term memory (LSTM) [101] and gated recurrent unit (GRU) [102] are variants of RNNs designed to learn long-term dependencies in data. Fig. 2 shows the structures of the RNN and LSTM. LSTM has a more complex structure than an RNN and includes three gates: forget, input, and output. The LSTM computational process involves numerous MAC operations and uses various parameters. Because the output of one cell is used as the input for the next cell, significant data movement is required. In summary, the operations within an RNN are performed sequentially, making the overall workflow challenging to parallelize. Furthermore, given various weight parameters, the value of one cell block is reused and its operation is repeated,

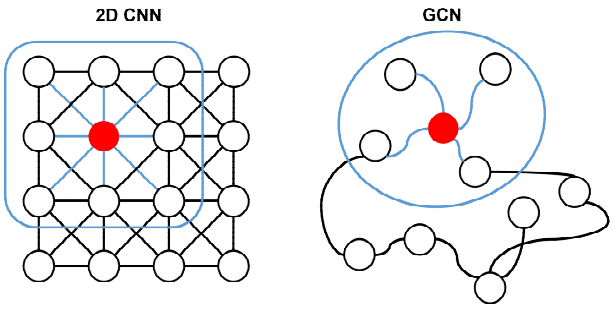


Fig. 1. 2D (two-dimensional) convolutional neural networks and graph convolution networks.

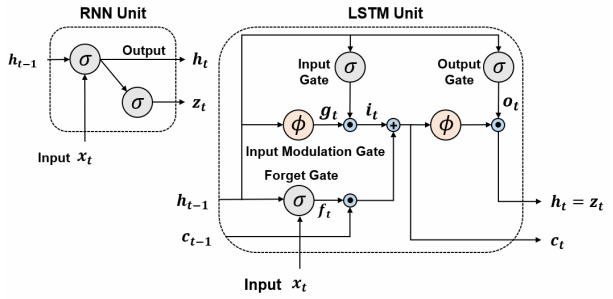


Fig. 2. RNN and LSTM architectures.

leading to constant data movement. To improve this situation, it is necessary to carefully consider the mapping of the data required for computation and manage the data flow effectively.

4) Transformer

Transformer [105] is a model that significantly outperforms the performance of RNNs in terms of processing sequence data. It consists of an encoder-decoder structure that incorporates a self-attention mechanism. The model is capable of training on sequence data more comprehensively and learning long-term dependencies effectively owing to its ability to process all elements in parallel. The more these encoder-decoder layers are stacked, the better the model understands the complex patterns and interrelationships among the embeddings. Because of these characteristics, transformers outperform RNNs, particularly in the field of natural language processing (NLP) [109]. Furthermore, owing to their scalability and flexibility, variants without decoders, such as the vision transformer (ViT) [108], have been introduced into the field of computer vision [114], where they demonstrate excellent performance.

Fig. 3 shows the structure of the transformer. The transformer consists of three main components: an

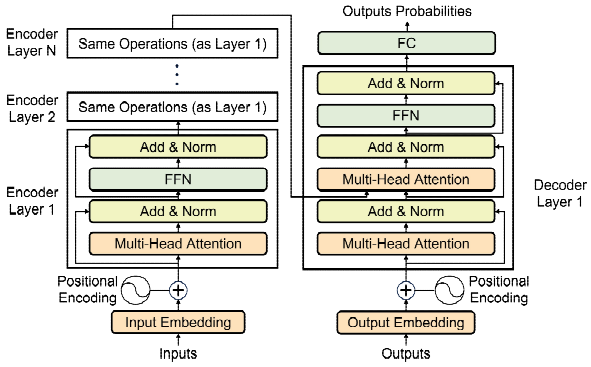


Fig. 3. Transformer architecture.

embedding layer, a multihead self-attention (MHA), and a feed-forward network (FFN). The embedding layer converts each sequence element into a continuous high-dimensional vector, thereby allowing the model to consistently extract complex information from each element and facilitate deeper learning. Transformer operations are primarily concentrated in MHA and FFN. MHA maintains interdependencies by training different feature maps from multiple attention heads and combining them into the output. Each attention head obtains a query (Q), key (K), and value (V) through matrix multiplication of weights. This process, which involves the parallel operation of different weights, requires high memory bandwidth and frequent data transfers. Self-attention is then applied through Q, K, and V, and the attentional head is updated. The FFN serves to mix the attention from the heads obtained through the MHA, preventing bias towards any single attention value. In summary, the transformer processes the inputs based on various weights and continually reuses the outputs as inputs for subsequent operations. Therefore, to optimize the operational process of the transformer, data mapping must be carefully considered so that the input/output values and weights can be effectively reused.

III. PIM FOR DEEP NEURAL NETWORKS

1. Technologies and Representative Architectures Needed for PIM

PIM fundamentally offers high throughput because it minimizes data transfer with the host processor by integrating data processing logic directly into memory, thus resolving the associated bottleneck [28, 29]. In the

DNN inference process, the most frequently performed MAC operations are executed in the PIM core to achieve high energy efficiency. In addition, during the DNN training process, PIM can reduce both processing time and power consumption by performing the computations necessary for weight updates directly within the memory [88, 94]. However, not all functions benefit from the application of PIM. For instance, it can be burdensome to process functions with high computational complexity and memory reusability using in-memory logic. Therefore, to determine where a specific function should be computed, it is necessary to establish appropriate metrics and analyze them using a benchmark simulator. DAMOV [30] is a memory simulator comprising a frequently used ramulator [31] and a zsim CPU simulator [32]. It extracts memory traces for each workload [117] using an Intel VTune profiler [33]. The extracted traces calculate the temporal/spatial locality and divide the causes of memory bottlenecks into six classes using three indicators: the last-to-first miss-ratio (LFMR), last-level-cache misses per kilo-instruction (LLC MPKI), and arithmetic intensity. Moreover, by conducting an experimental analysis with 77 K functions, we demonstrated its reliability and applicability across various research areas.

Current PIM research is largely categorized into commercially accessible DRAM-based PIM research [52-59, 81-85] and research utilizing next-generation memory [90-99], both of which are presented in a competitive manner. Unlike academic research, mass-producible PIM products fundamentally utilize bank-level parallelism of DRAM for computation processing. In addition, they also consider maximizing compatibility with existing mass-produced products and prioritizing cost aspects, such as minimizing the area occupied by logic operations and addressing heat-dissipation issues. The HBM-PIM [58] is an addition of PIM functionality to the HBM architecture, designed to increase memory bandwidth and energy efficiency by performing computational processing within the memory. It proposes not only a hardware architecture but also a software stack. The software stack supports FP16 operations, MAC, general matrix-matrix product (GEMM), and activation functions, along with the operation logic loaded onto the HBM by applying the LUT. In addition, it allows programmers to write PIM microkernels using PIM commands to maximize performance. The hardware architecture was implemented based on 20 nm DRAM technology and integrated with an unmodified commercial processor to prove its practicality and effectiveness at the system level. Furthermore, it is designed to be replaceable because it is compatible with existing HBM. By implementing the proposed PIM architecture, there was a significant improvement in the performance of memory-bound neural network kernels and applications. Specifically, the performance of neural network kernels increased by 11.2×, while applications showed a 3.5× improvement. Additionally, the energy consumption per bit transfer was reduced by 3.5×, resulting in an overall enhancement of the system's energy efficiency by 3.2× when running applications.

Newton’s architecture [59] was designed as an accelerator in memory (AiM) for DNNs. In this design, a minimum number of computing units were placed in the DRAM to satisfy the area constraints, which can be a problem in the hardware design for PIM. The computing units included MAC operations and buffers. It also uses an interface similar to that of DRAM so that the host can issue commands for PIM computing. The PIM matches the internal DRAM bandwidth and speed, captures input reuse, and uses a global input vector buffer to divide the buffer area costs across all channels. The three optimization techniques proposed by Newton helped the PIM-host interface overcome bottlenecks: 1) Grouping multiple computational tasks in the in-bank and bank groups. 2) Support complex, multistep computing commands to process multiple stages of operations simultaneously. 3) The strength of the internal low-dropout (LDO) regulator and DC-DC pump driver should be increased to allow for higher current and faster voltage recovery. As a result, Newton applied to HBM2E achieves an average speed improvement of 10× over a system assumed to ideally use the external DRAM bandwidth without applying PIM and 54× over a GPU.

The UPMEM PIM architecture [52] was the first commercialized PIM architecture that combined traditional DRAM memory arrays and a common instruction sequence core: the DRAM processing unit (DPU). DPUs are a concept proposed for UPMEM and are used to perform operations within memory chips. The DPU has exclusive access to a 64MB DRAM bank,

known as the main random-access memory (MRAM), 24 KB of command memory, and 64 KB of scratchpad memory, called the working random access memory (WRAM). This allows programmers to write code to be executed on the DPU and process the data within the memory. This implies that the data transfer between the host processor and DPU can be controlled, allowing for the selection of parallel and sequential processing.

On the other hand, the most commonly used next-generation memory in PIM architecture is ReRAM [36, 39, 91, 95, 100, 103, 107]. The ReRAM crossbar array consists of cells arranged in rows and columns. This array can be used for memory purposes and can efficiently perform computations such as the general matrix-vector product (GEMV), composed of MAC operations. In addition, the use of a crossbar array can significantly reduce the overhead and energy related to memory movement. In particular, as a pioneering study on ReRAM-based PIM, PRIME [91] distinguished the internal array space of a bank as memory a subarray (MemS), full function subarray (FFS), and buffer subarray. MemS is a circuit that stores only data. FFS allows the crossbar to be used for both memory and operation logic, achieving minimum area overhead. To enable this, multiple voltage sources are added to provide an accurate input voltage. The column multiplexer provides an analog subtraction unit and a nonlinear threshold value unit, and the SA is modified to achieve high precision.

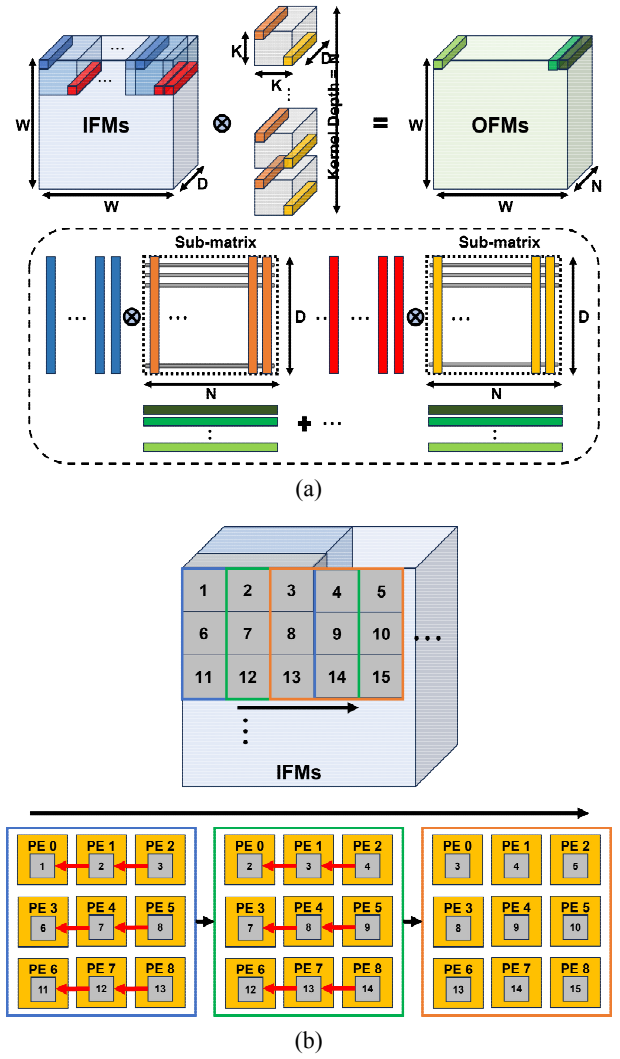


Fig. 4. Processing-in-Memory for CNN proposed in [45]: (a) A basic mapping method of input and weight data, with kernel moving in multiple cycles; (b) An example of IFMs transferred among PEs and how the kernel slides over the input.

2. PIM for CNN

Numerous PIM studies primarily support the MAC operation required by CNN [46, 47, 52-59]. However, this study focuses on PIM research that employs data mapping methods and dataflow necessary for CNN operations. Efficient data handling in the CNN inference process is crucial, with particular emphasis on maximizing the reuse of weights as well as the input and output feature maps used between layers.

1) Inference Phase

Peng *et al.* [45] proposed an ReRAM-based PIM accelerator that adapted the data-mapping technique proposed by Fey *et al.* [44] for the CONV layer. This reduces the use of interconnects and buffers by reusing the input data and weights. As shown in Fig. 4(a), a 3D kernel of size K×K×D is arranged in vertical columns, and the input feature map (IFM) is arranged in a similar manner in K×K submatrices within 1×1×D kernels. As shown in Fig. 4(b), computation of the subarrays proceeds as a single PE within the ReRAM subarray. This method is designed to maximize the reuse of IFMs and weights as the kernel (*i.e.,* weights) slides over them during computation. Consequently, this study achieved a 2.1× increase in speed and 17% improvement in energy efficiency (measured in TOPS/W) during the inference phase with the VGG-16 model compared with [92].

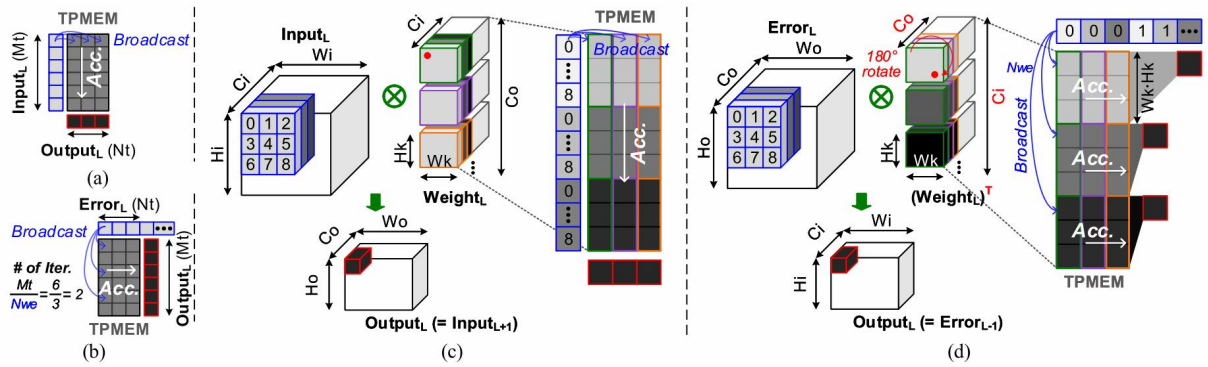


Fig. 5. Data mapping of T-PIM: (a) FWP layer; (b) BWP layer; (c) FWP, CONV layer; (d) BWP, CONV layer (Reprinted from [88] with permission).

2) End-to-End Training Phase

Backpropagation in CNNs requires a significant amount of computation because it involves computing the gradients for each layer and updating the weights to train the model. It is considered memory-bound because it includes storing and tracking the intermediate features and gradients of all the layers, which is more intensive than inferring the CONV layer. Therefore, higher efficiency can be expected by optimizing the training process in the PIM.

T-PIM [88] is a DRAM-based PIM study considering the end-to-end training of CNN models. Fig. 5 represents the data mapping of T-PIM that reduces the overhead caused by data rearrangement in DRAM and optimizes the data access to weights. Fig. 5(a) and (b) show the data mapping methods during the forward pass (FWP) and backward pass (BWP) within the MLP layer, respectively. To maximize the utilization of DRAM's cell array without rearranging data, the size of the tile is set to $M_t \times N_t$ and each weight is mapped to DRAM's column addresses. During the FWP process, the input vector is flattened to size M_t (Input$_L$ (M_t)) and multiplied with the weights arranged in DRAM. Each column is then accumulated into an output buffer of size N_t (Output$_L$ (N_t)). For the BWP process, to use the weights aligned in the FWP process without additional rearrangement, the loss (Error$_L$ (N_t)) is flattened into N_t elements and performs vector operations with the weights. Each row is then accumulated into an output buffer of size M_t (Output$_L$ (M_t)). Fig. 5(c) and (d) represent the data mapping methods used during the FWP and BWP in the CONV layer, respectively. Similar to the MLP layer, weights (Weight$_L$) are arranged to column addresses by kernel size ($=Wk \times Hk$), so the weights can be reused without the need for data rearrangement. T-PIM shows high efficiency of 0.84-7.59 TOPS/W for 8-bit input data and 0.25-2.21 TOPS/W for 16-bit input data in VGG16 model training, using the non-zero computing, powering off computing method.

3. PIM for GCN

The processing steps of a GCN (*e.g.,* aggregation, combination, embedding, message passing, and readout) are mostly low in operational complexity, data-dependent, and performed repetitively. Among these, aggregation must process large amounts of data to combine the information of each node with that of its neighboring nodes. Moreover, these operations have the characteristic that they must be performed as different operation combinations depending on the relationship between each node and its neighbors. These characteristics require a large amount of computation and high memory bandwidth. Therefore, these drawbacks can be effectively mitigated using PIM. The PIM for GCN has also been approached by actively utilizing an ReRAM crossbar to perform operation processing as an analog computing method [36].

Two representative techniques are the MAC crossbar and content addressable memory (CAM) crossbar [37]. Among these two, the CAM crossbar performs content-based searches. This allows a parallel associative search by broadcasting the search key across multiple rows.

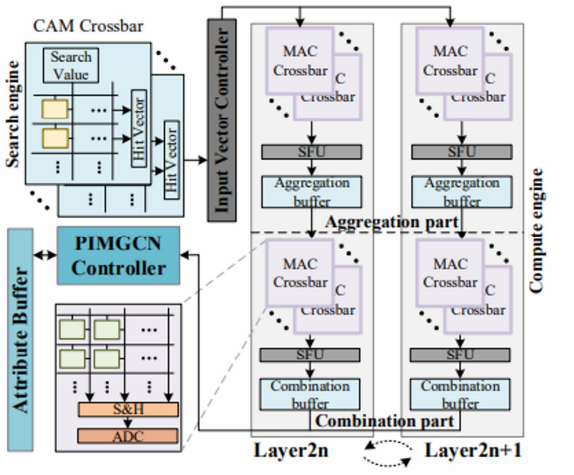


Fig. 6. PIMGCN architecture overview (Reprinted from [39] with permission).

This enabled the storage of more data on a chip in the same area. It was shown in TCAM [38] that 2-transistor-2-resistor ReRAM can achieve 3× higher density than the existing 8-transistor SRAM. The MAC crossbar can effectively perform the VMM with low energy consumption through bit-line current accumulation. This process can be described in three steps. 1) The elements of the matrix were converted to voltage and assigned to the crossbar, and the resistor of the cell was precisely adjusted to correspond to the elements. 2) The vector is converted to a voltage, which is accumulated on the word line. 3) The current of the bit line was measured, and the sum of the currents of all cells connected to the bit line was obtained as the product of the column and vector.

Fig. 6 shows the overall architecture of PIM-GCN [39], which consists of a central controller, a search engine, and two computing engines. Each of these comprises a CAM crossbar and a MAC crossbar, and the two computing engines operate in a typical ping-pong architecture, alternately performing aggregation and combination. The central controller initially loaded the graph data and finally exported the GCN results back to the external DRAM. It also generates the necessary control logic for the CAM crossbar, the MAC crossbar, and the special function unit (SFU). The SFU, composed of a shift-and-add (S&A) unit and scalar arithmetic and logic (sALU) units, processes the partial results derived from the MAC crossbar. PIM-GCN introduces not only a hardware architecture that can maximize inter-vertex parallelism, but also a technique for optimizing node grouping without violating independence, providing scheduling for these groups to operate independently at each layer. It also proposes a timing strategy to reduce idle time owing to differences in read/write latency.

GCIM [40] is an accelerator research that presents a software-hardware co-design approach, becoming the first to enable efficient data processing of GCNs in 3D stack memory. From a hardware design perspective, the GCIM proposes a logic-in-memory (LIM) die that integrates light computing units near the DRAM bank, fully utilizing the bandwidth and parallelism at the bank level. The GCIM offloads memory-bound aggregation operations onto the LIM die. Each LIM bank group is equipped with an LLU consisting of a MAC array, vertex feature buffer (VFB), look-ahead FIFO, CAM, and a controller to accelerate the aggregation phase. A MAC array was used to execute the aggregation operations. VFB is used to buffer the output features during the aggregation phase. Look-ahead FIFO is a special edge buffer implemented as a scratch-pad memory that processes the frontmost edge upon receiving a signal from the controller. The CAM provides key-value storage that records the ID of nonlocal vertices and the local addresses where their replicas are buffered. The controller is a data-based control unit that processes the aggregation operations of local vertices. On the software side, GCIM proposes a data-mapping algorithm that considers locality. It balances the workload by splitting the input graph into subgraphs considering the connection strength of the nodes. Here, if the weight between two vertices is large or if multiple paths exist, the strength is determined to be strong. The divided subgraphs are assigned to the vault and mapped to the LIM bank group. This was optimized to utilize a high bandwidth and reduce unnecessary data movement. This significantly improves the computational efficiency while preventing redundant calculations. In addition, the GCIM adopts a sequential mapping strategy to maximize data locality and minimize the processing delay of the aggregation. This optimization technique uses dynamic programming [41], a mechanism that saves the optimal solution of a subproblem and reuses it to determine the optimal solution of the entire problem. Based on experimental results, GCIM demonstrated a remarkable improvement in inference speed compared to other models. Specifically, it achieved a speed enhancement of

580.02× compared to HyGCN [42], 275.37× compared to CIM-HyGCN, and 272.01× compared to PyG-CPU [43]. These results highlight the significant performance boost offered by GCIM in terms of inference speed.

Although the two studies mentioned earlier were based on different memory-based PIM hardware architectures, they both proposed algorithms for grouping and mapping graph data nodes in a memory-friendly manner, and effectively handled GCN aggregation and combination operations.

4. PIM for RNN

RNN and LSTM structures can be effectively applied with PIM owing to their similarity to CONV layers and their ability to reuse feature maps and weights. ERA-LSTM [103] is a PuM architecture that uses ReRAM crossbars. It optimized the RNN's weight precision and digital-to-analog converter (DAC) in Long *et al.* [100] PIM architecture and applied systolic dataflow to improve computing efficiency and performance. Fig. 7(a) shows the overall structure of ERA-LSTM. The VMM unit in Fig. 7(b) stores the weights of the four LSTM gates and uses a digital-to-analog converter to deliver the input data and hidden states from the I/O buffer to the analog ReRAM crossbar. The computational results of the VMM unit are transmitted to an element-wise (EW) unit. The EW unit enables EW operation of the LSTM cell in the three feedforward layers. In addition, the VMM and EW units efficiently handle each of the four gate weights $(e.g., W_f, W_i, W_g, W_o)$ by splitting each weight into four weights $(e.g., W_{00} - W_{11})$ and tiling each weight into a tile for computation. In addition, the NN operation used an approximator to minimize the overhead caused by analog-to-digital converters, achieving a 6.1× operational efficiency compared with Long *et al.* [100].

PSB-RNN [104] is another PuM architecture that uses a ReRAM crossbar. PSB-RNN transforms the MAC operation required for the RNN model into a single weights matrix using Fast Fourier Transform (FFT). The real (Re) and imaginary (Im) components of the resulting matrix are mapped onto the ReRAM crossbar, thereby enabling the retrieval of complex number operation results from each PE result. This method

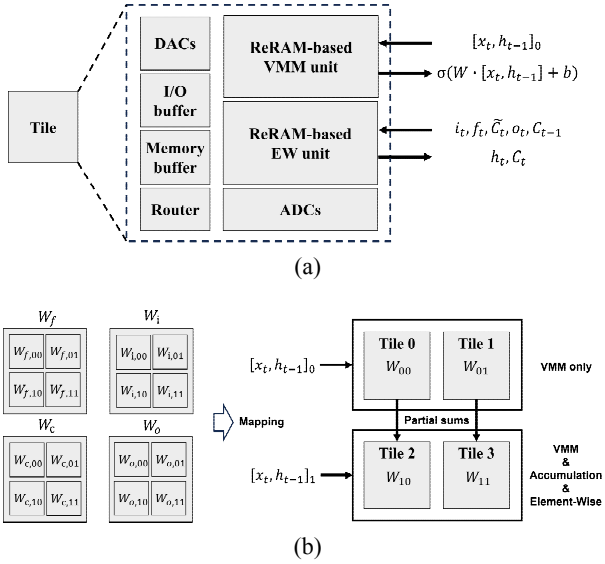
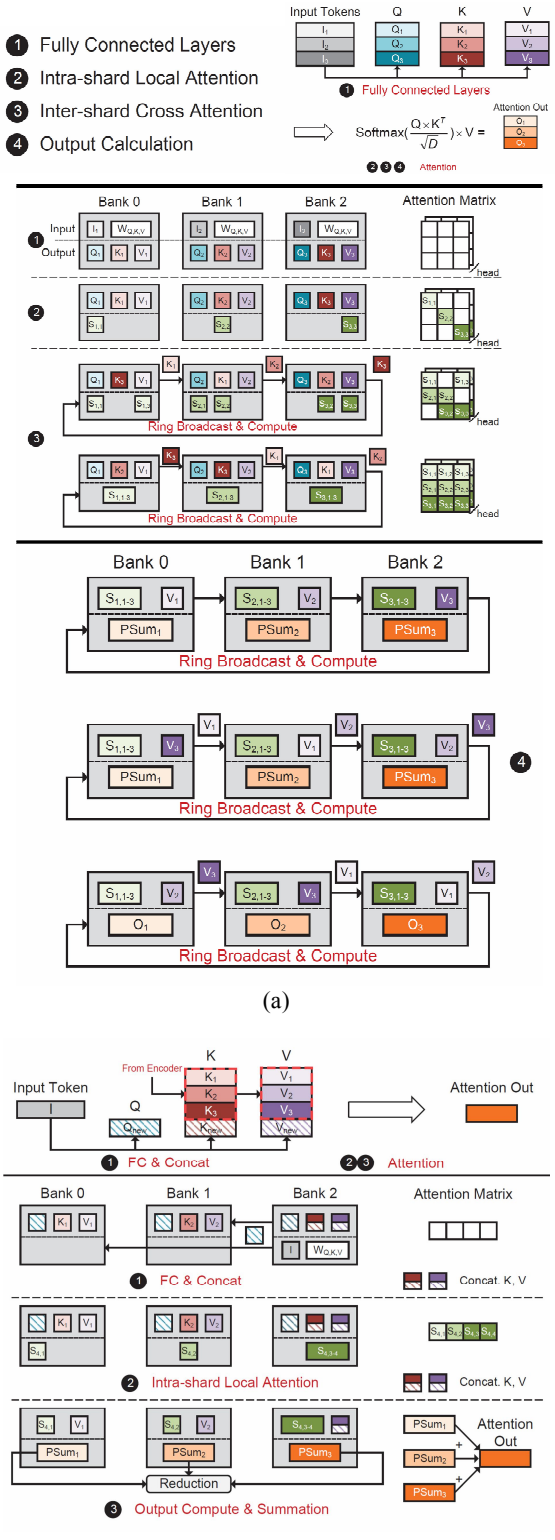


Fig. 7. ERA-LSTM: (a) architecture overview; (b) Mapping a LSTM cell to multiple tiles.

yielded a computational efficiency that was 17× higher than that of Long *et al.* [100] for the LSTM model. Although this study requires additional operations and tasks beyond data mapping for the traditional LSTM model, it proposes an effective method for ReRAM crossbar PIM by mapping data for a complex number of operations necessary for MAC and utilizing the data flow.

5. PIM for Transformer

TransPIM [106] is an HBM-based PnM designed for efficient transformer utilization. An arithmetic control unit (ACU) was allocated to each bank for computation, and a token-based data shading scheme was proposed to allow parallel processing by dividing and assigning the data required for the calculation to the HBM's bank stack. The study also optimizes data using a token-based transformer operation method, which enables independent operations between tokens, in contrast to the existing layer-operated transformer structure. Fig. 8(a) illustrates the encoder process of TransPIM. The input token size is L×D, where L signifies the number of tokens and D indicates the size of the embedding vector's dimension. Input tokens I_1, I_2 and I_3 are allocated to each bank using a technique that distributes each input token to N banks. Based on this, the embedding values Q_i, K_i, V_i corresponding to each input token are calculated and assigned to the same bank, followed by a



(a)

(b)

Fig. 8. Token-based data sharding scheme and the dataflow of Transformer: (a) encoder; (b) decoder in TransPIM (Reprinted from [106] with permission).

self-attention operation. For the MHA, K_i and V_i are sequentially transferred to bank i +1 and sent to another bank for calculation using the ring-broadcast technique, thus enabling computation with minimal data transmission between banks. Fig. 8(b) shows a decoder block, where K and V are received from the encoder vector for reuse, and only the last bank obtains new Q, K, and V vectors for the fully connected layer computation. The new Q_{new} is broadcast to all other banks to calculate the attention score, and K_{new} and V_{new} are concatenated with the previous K_i and V_i of the last bank. Each bank stores the weights for Q, K, and V during this time, and the ring broadcast technique is employed to reuse the stored weights and Q, K, and V values in the other banks, facilitating the efficient processing of repeated NN operations. To this end, this study incorporates the ACU onto the banks of HBM memory and adds a ring broadcast unit between the banks. This allows for a reduction of more than 30.8% in the data movement overhead on average compared with the existing transformer, with only 4% additional area overhead relative to the original DRAM. This study ensured that the PIM power remained below the DRAM power budget of 60 W.

ReTransformer [107] proposed and applied optimization techniques to effectively accelerate GEMV operations within the transformer inference process, and softmax is suitable for low-power implementation in ReRAM-based PIM. This study has a similar direction to the existing ReRAM-based transformer-based workload target PIM study implementing MatMul operations inside ReRAM and applying optimization techniques. Thus, the latency of the operational process can be reduced. Specifically, this paper proposes a method of decomposing the operation into two consecutive multiplication steps to solve the compute-write-compute dependency that occurs when implementing the MatMul operation between Q and K^T during the transformer inference process using ReRAM. Consequently, the latency recorded in the crossbar of the ReRAM can be eliminated. In addition, a modified hybrid softmax formula that can maximize the crossbar arrangement of the ReRAM was proposed and applied to the softmax operation; as a result, only 0.691 mW was consumed and implemented, unlike 1.023 mW for the existing softmax

operation. Finally, this study achieved a 23.21× computational efficiency improvement and a 1,086× power consumption reduction compared to NVIDIA TITAN RTX GPUs.

6. Discussions

The PIM is a new architecture that integrates processing and memory units into the PE, thereby enabling efficient data processing. However, owing to the integration of computational functions into memory cells, PIM may be limited in handling complex operations, and can cause performance degradation when computationally intensive operations are required. Moreover, PIM's complex control structure and limited memory capacity pose limit the full and effective handling of increasingly large AI workloads. For PIM cores to be effectively applied to AI workloads, clear criteria are required to determine whether operands should be computed in the host processor or the PIM core. These criteria are typically derived by statistically analyzing the results measured at the functional level using benchmark simulators [34, 35]. In addition, in the PIM design process, the mapping of processes and parameters, as well as the data flow considering complex operations, must be carefully incorporated. In previous PIM studies, these considerations were designed heuristically. However, with increasingly diverse PIM architectures and algorithms, there is an urgent need for research on compilers that can automatically optimize workload functions, data mapping, and data flow.

IV. CONCLUSIONS

In this paper, we present industrial products and analyze previous studies on PIM architectures for DNNs. We provide detailed explanations of the characteristics of representative DNNs and the overall structure of PIM. Specifically, we delve into how processing techniques, data mapping, and data flow are handled in PIM from a DNN perspective, drawing insights from representative studies. Additionally, we highlight the challenges that need to be addressed for PIM to replace the traditional von Neumann architecture. We emphasize the importance of conducting research on a universal software stack and compiler to effectively implement existing studies. We anticipate that this study will offer valuable insights to researchers exploring PIM technology.

ACKNOWLEDGMENTS

This study was supported by the Research Program funded by the SeoulTech (Seoul National University of Science and Technology).

REFERENCES

[1] D. Kreuzberger, N. Kühl, and S. Hirschl, “Machine learning operations (mlops): Overview, definition, and architecture,” *IEEE Access*, vol. 11, pp. 31866-31879, 2023.

[2] W. Liu *et al*., “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11-26, 2017.

[3] J. A. Suykens and J. Vandewalle, “Training multilayer perceptron classifiers based on a modified support vector method,” *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 907-911, 1999.

[4] J. -W. Jang *et al.*, “Sparsity-aware and re-configurable NPU architecture for Samsung flagship mobile SoC,” in *Proc. 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 15-28.

[5] A. Reuther *et al*., “AI accelerator survey and trends,” in *Proc. 2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, pp. 1-9.

[6] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, “Memory devices and applications for in-memory computing,” *Nature nanotechnology*, vol. 15, no. 7, pp. 529-544, 2020.

[7] M. Imani, S. Gupta, and T. Rosing, “Ultra-efficient processing in-memory for data intensive applications,” in *Proc. 54th Annual Design Automation Conference 2017*, 2017, pp. 1-6.

[8] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” *Neural networks for perception*, Elsevier, 1992, pp. 65-93.

[9] A. Ganguly, R. Muralidhar, and V. Singh, “Towards Energy Efficient non-von Neumann Architectures for Deep Learning,” in*2Proc. 20th*

International Symposium on Quality Electronic Design (ISQED), 2019, pp. 335-342.

[10] J. Li *et al.*, "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators," in *Proc. 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 343-348.

[11] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova, "Nonlinear approximation and (deep) ReLU networks," *Constructive Approximation*, vol. 55, no. 1, pp. 127-172, 2022.

[12] T. Liu, T. Qiu, and S. Luan, "Hyperbolic-tangent-function-based cyclic correlation: Definition and theory," *Signal Processing*, vol. 164, pp. 206-216, 2019.

[13] J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *From Natural to Artificial Neural Computation: International Workshop on Artificial Neural Networks Malaga-Torremolinos*, 1995, pp. 195-201.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84-90, 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE conference on computer vision and pattern recognition*, 2016, pp. 770-778.

[16] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.

[17] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE Transactions on neural networks and learning systems*, vol. 33, no. 12, pp. 6999-7019, Dec. 2022.

[18] J. Yin *et al.*, "Highly parallel GEMV with register blocking method on GPU architecture," *Journal of Visual Communication and Image Representation*, vol. 25, no. 7, pp. 1566-1573, 2014.

[19] J. Cheng *et al.*, "Cache-Major: A Hardware Architecture and Scheduling Policy for Improving DRAM Access Efficiency in GEMV," in *Proc. 2022 IEEE 16th International Conference on Solid-State & Integrated Circuit Technology (ICSICT)*, 2022, pp. 1-3.

[20] S. Khan *et al.*, "Transformers in vision: A survey," *ACM computing surveys (CSUR)*, vol. 54, no. 10s, pp. 1-41, 2022.

[21] P. C. Chen and Y. K. Hwang, "SANDROS: a dynamic graph search algorithm for motion planning," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 3, pp. 390-403, 1998.

[22] K. M. Chandy and J. Misra, "Distributed computation on graphs: Shortest path algorithms," *Communications of the ACM*, vol. 25, no. 11, pp. 833-837, 1982.

[23] E. M. Palmer, "On the spanning tree packing number of a graph: a survey," *Discrete Mathematics*, vol. 230, no. 1-3, pp. 13-21, 2001.

[24] C. C. Aggarwal and H. Wang, "A survey of clustering algorithms for graph data," *Managing and mining graph data*, pp. 275-301, 2010.

[25] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4-24, 2020.

[26] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: a comprehensive review," *Computational Social Networks*, vol. 6, no. 1, pp. 1-23, 2019.

[27] M. Chen, Z. Wei, Z. Huang, B. Ding, and Y. Li, "Simple and deep graph convolutional networks," in *Proc. International conference on machine learning*, 2020, pp. 1725-1735.

[28] X. Zou, S. Xu, X. Chen, L. Yan, and Y. Han, "Breaking the von Neumann bottleneck: architecture-level processing-in-memory technology," *Science China Information Sciences*, vol. 64, no. 6, p. 160404, 2021.

[29] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data movement is all you need: A case study on optimizing transformers," in *Proceedings of Machine Learning and Systems*, vol. 3, pp. 711-732, 2021.

[30] G. F. Oliveira *et al.*, "DAMOV: A new methodology and benchmark suite for evaluating data movement bottlenecks," *IEEE Access*, vol. 9, pp. 134457-134502, 2021.

[31] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45-49, 2015.

[32] D. Sanchez and C. Kozyrakis, "ZSim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475-486, 2013.

[33] C.-K. Luk *et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, vol. 40, no. 6, pp. 190-200, 2005.

[34] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE computer architecture letters*, vol. 10, no. 1, pp. 16-19, 2011.

[35] J. D. Leidel and Y. Chen, "Hmc-sim: A simulation framework for hybrid memory cube devices," *Parallel Processing Letters*, vol. 24, no. 4, p. 1442002, 2014.

[36] S. Mittal, "A survey of ReRAM-based architectures for processing-in-memory and neural networks," *Machine learning and knowledge extraction*, vol. 1, no. 1, pp. 75-114, 2018.

[37] N. Challapalle *et al.*, "GaaS-X: Graph analytics accelerator supporting sparse data representation using crossbar architectures," in *Proc. 2020 ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 433-445.

[38] R. Yang *et al.*, "Ternary content-addressable memory with MoS2 transistors for massively parallel data search," *Nature Electronics*, vol. 2, no. 3, pp. 108-114, 2019.

[39] T. Yang *et al.*, "PIMGCN: A ReRAM-based PIM design for graph convolutional network acceleration," in *Proc. 2021 58th ACM/IEEE Design Automation Conference*, 2021, pp. 583-588.

[40] J. Chen *et al.*, "GCIM: Toward Efficient Processing of Graph Convolutional Networks in 3D-Stacked Memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 3579-3590, Nov. 2022.

[41] O. A. Alzubi *et al.*, "An optimal pruning algorithm of classifier ensembles: dynamic programming approach," *Neural Computing and Applications*, vol. 32, pp. 16091-16107, 2020.

[42] M. Yan et al., "Hygcn: A gcn accelerator with hybrid architecture," in *Proc. 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 15-29.

[43] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[44] T. Gokmen, M. Onen, and W. Haensch, "Training deep convolutional neural networks with resistive cross-point devices," *Frontiers in neuroscience*, vol. 11, p. 538, 2017.

[45] X. Peng, R. Liu, and S. Yu, "Optimizing weight mapping and data flow for convolutional neural networks on RRAM based processing-in-memory architecture," in *Proc. 2019 IEEE International Symposium on Circuits and Systems*, 2019, pp. 1-5.

[46] A. Roohi, S. Angizi, D. Fan, and R. F. DeMara, "Processing-in-memory acceleration of convolutional neural networks for energy-effciency, and power-intermittency resilience," in *Proc. 20th International Symposium on Quality Electronic Design (ISQED)*, 2019, pp. 8-13.

[47] Y. Wang, W. Chen, J. Yang, and T. Li, "Towards memory-efficient allocation of CNNs on processing-in-memory architecture," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 6, pp. 1428-1441, 2018.

[48] H. Dbouk, S. K. Gonugondla, C. Sakr, and N. R. Shanbhag, "A 0.44-μJ/dec, 39.9-μs/dec, recurrent attention in-memory processor for keyword spotting," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 7, pp. 2234-2244, 2020.

[49] A. K. Ramanathan *et al.*, "Look-up table based energy efficient processing in cache support for neural network acceleration," in *Proc. 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 88-101.

[50] M. He *et al.*, "Newton: A DRAM-maker's accelerator-in-memory (AiM) architecture for machine learning," in *Proc. 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372-385.

[51] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer," in *Proc. 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 1071-1085.

[52] J. Gómez-Luna *et al.*, "Benchmarking memory-centric computing systems: Analysis of real processing-in-memory hardware," in *Proc. 2021*

12th International Green and Sustainable Computing Conference (IGSC), 2021, pp. 1-7.

[53] F. Devaux, "The true processing in memory accelerator," in *Proc. 2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1-24.

[54] H. Shin *et al*. "McDRAM: Low latency and energy-efficient matrix computations in DRAM," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2613-2622, 2018.

[55] S. Cho, H. Choi, E. Park, H. Shin, and S. Yoo, "McDRAM v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge," *IEEE Access*, vol. 8, pp. 135223-135243, 2020.

[56] Y. C. Kwon *et al.*, "25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications," in *Proc. 2021 IEEE International Solid-State Circuits Conference,* Feb. 2021, pp. 350-352.

[57] S. Lee *et al.*, "Hardware architecture and software stack for PIM based on commercial DRAM technology: Industrial product," in *Proc. 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43-56.

[58] J. H. Kim *et al.*, "Aquabolt-XL: Samsung HBM2-PIM with in-memory processing for ML accelerators and beyond," in *Proc. 2021 IEEE Hot Chips 33 Symposium (HCS)*, 2021, pp. 1-26.

[59] D. Kwon *et al.*, "A 1ynm 1.25 V 8Gb 16Gb/s/Pin GDDR6-Based Accelerator-in-Memory Supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep Learning Application," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 291-302, 2022.

[60] A. Boroumand *et al.*, "LazyPIM: Efficient support for cache coherence in processing-in-memory architectures," *arXiv preprint arXiv:1706.03162*, 2017.

[61] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis, "Tetris: Scalable and efficient neural network acceleration with 3d memory," in *Proc. 22nd International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017, pp. 751-764.

[62] M. P. Drumond Lages De Oliveira *et al.*, "The Mondrian data engine," in *Proc. 44th International Symposium on Computer Architecture*, 2017.

[63] G. Dai *et al.*, "Graphh: A processing-in-memory architecture for large-scale graph processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640-653, 2018.

[64] M. Zhang *et al.*, "GraphP: Reducing communication for PIM-based graph processing with efficient data partition," in *Proc. 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 544-557.

[65] Y. Huang *et al.*, "A heterogeneous PIM hardware-software co-design for energy-efficient graph processing," in *Proc. 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 684-695.

[66] Y. Zhuo et al., "Graphq: Scalable pim-based graph processing," in *Proc. 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 712-725.

[67] J. Gómez-Luria *et al.*, "Machine Learning Training on a Real Processing-in-Memory System," in *Proc. 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 292-295.

[68] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "Towards efficient sparse matrix vector multiplication on real processing-in-memory systems," *arXiv preprint arXiv:2204.00900*, 2022.

[69] I. Fernandez *et al.*, "Exploiting near-data processing to accelerate time series analysis," in *Proc. 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 279-282.

[70] G. F. Oliveira, A. Boroumand, S. Ghose, J. Gómez-Luna, and O. Mutlu, "Heterogeneous Data-Centric Architectures for Modern Data-Intensive Applications: Case Studies in Machine Learning and Databases," in *Proc. 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2022, pp. 273-278.

[71] A. C. Jacob et al., "Compiling for the active memory cube," Tech. rep. RC25644 (WAT1612-008). IBM Research Division, Tech. Rep., 2016.

[72] S. Lloyd and M. Gokhale, "Design space exploration of near memory accelerators," in *Proc. International Symposium on Memory Systems*, 2018, pp. 218-220.

[73] M. Gokhale, S. Lloyd, and C. Hajas, "Near memory data structure rearrangement," in *Proc. Int. Symp. on Memory Systems*, 2015, pp. 283-290.

[74] S. Lloyd and M. Gokhale, "In-memory data rearrangement for irregular, data-intensive computing," *Computer*, vol. 48, no. 8, pp. 18-25, 2015.

[75] A. Rodrigues, M. Gokhale, and G. Voskuilen, "Towards a scatter-gather architecture: hardware and software issues," in *Proc. International Symposium on Memory Systems*, 2019, pp. 261-271.

[76] S. Lloyd and M. Gokhale, "Near memory key/value lookup acceleration," in *Proc. International Symposium on Memory Systems*, 2017, pp. 26-33.

[77] J. Landgraf, S. Lloyd, and M. Gokhale, "Combining emulation and simulation to evaluate a near memory key/value lookup accelerator," *arXiv preprint arXiv:2105.06594*, 2021.

[78] R. Nair, "Evolution of memory architecture," *Proc. IEEE*, vol. 103, no. 8, pp. 1331-1345, 2015.

[79] L. Ke *et al.*, "Near-Memory Processing in Action: Accelerating Personalized Recommendation With AxDIMM," *IEEE Micro*, vol. 42, no. 1, pp. 116-127, 2022.

[80] D. Lee *et al.*, "Improving in-memory database operations with acceleration DIMM (AxDIMM)," in *Proc. Data Management on New Hardware*, 2022, pp. 1-9.

[81] S. Li *et al.*, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proc. 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 288-301.

[82] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "DrAcc: A DRAM based accelerator for accurate CNN inference," in *Proc. annual design automation conference*, 2018, pp. 1-6.

[83] S. Li *et al.*, "Scope: A stochastic computing engine for dram-based in-situ accelerator," in *Proc. 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 696-709.

[84] X. Xin, Y. Zhang, and J. Yang, "ELP2IM: Efficient and low power bitwise operation processing in DRAM," in *Proc. 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 303-314.

[85] V. Seshadri *et al.*, "Ambit: In-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proc. Annual IEEE/ACM International Symposium on Microarchitecture*, 2017, pp. 273-287.

[86] M. Ali *et al.*, "IMAC: In-memory multi-bit multiplication and accumulation in 6T SRAM array," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2521-2531, 2020.

[87] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733-1743, 2020.

[88] J. Heo, J. Kim, S. Lim, W. Han, and J.-Y. Kim, "T-PIM: An Energy-Efficient Processing-in-Memory Accelerator for End-to-End On-Device Training," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 3, pp. 600-613, March 2023.

[89] A. Biswas and A. P. Chandrakasan, "Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications," in *Proc. 2018 IEEE International Solid-State Circuits Conference-(ISSCC)*, 2018, pp. 488-490.

[90] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14-26, 2016.

[91] P. Chi *et al.*, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27-39, 2016.

[92] X. Sun, S. Yin, X. Peng, R. Liu, J. Seo, and S. Yu, "XNOR-RRAM: A scalable and parallel resistive synaptic architecture for binary neural networks," in *Proc. 2018 Design, Automation & Test in Europe Conference & Exhibition*, 2018, pp. 1423-1428.

[93] T. Tang, L. Xia, B. Li, Y. Wang, and H. Yang,

"Binary convolutional neural network on RRAM," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 782-787.

[94] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *Proc. International Symposium on Computer Architecture*, 2019, pp. 802-815.

[95] L. Song, X. Qian, H. Li, and Y. Chen, "Pipelayer: A pipelined reram-based accelerator for deep learning," in *Proc. IEEE international symposium on high performance computer architecture (HPCA)*, 2017, pp. 541-552.

[96] S. Angizi, Z. He, A. Awad, and D. Fan, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 1123-1136, 2019.

[97] A. D. Patil, H. Hua, S. Gonugondla, M. Kang, and N. R. Shanbhag, "An MRAM-based deep in-memory architecture for deep neural networks," in *Proc. 2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019, pp. 1-5.

[98] S. Angizi, Z. He, F. Parveen, and D. Fan, "IMCE: Energy-efficient bit-wise in-memory convolution engine for deep neural network," in *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2018, pp. 111-116.

[99] S. Angizi, Z. He, A. S. Rakin, and D. Fan, "Cmp-pim: an energy-efficient comparator-based processing-in-memory neural network accelerator," in *Proc. Annual Design Automation Conference*, 2018, pp. 1-6.

[100] Y. Long, T. Na, and S. Mukhopadhyay, "ReRAM-based processing-in-memory architecture for recurrent neural network acceleration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 12, pp. 2781-2794, 2018.

[101] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[102] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[103] J. Han, H. Liu, M. Wang, Z. Li, and Y. Zhang, "ERA-LSTM: An efficient ReRAM-based architecture for long short-term memory," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 6, pp. 1328-1342, 2019.

[104] N. Challapalle *et al.*, "Psb-rnn: A processing-in-memory systolic array architecture using block circulant matrices for recurrent neural networks," in *Proc. Design, Automation & Test in Europe Conference & Exhibition*, 2020, pp. 180-185.

[105] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[106] M. Zhou, W. Xu, J. Kang, and T. Rosing, "TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer," in *Proc. IEEE Int. Symp. on High-Performance Computer Architecture*, 2022, pp. 1071-1085.

[107] X. Yang, B. Yan, H. Li, and Y. Chen, "ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration," in *Proc. 39th International Conference on Computer-Aided Design*, 2020, pp. 1-9.

[108] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[109] S. M. Lakew, M. Cettolo, and M. Federico, "A comparison of transformer and recurrent neural networks on multilingual neural machine translation," *arXiv preprint arXiv:1806.06957*, 2018.

[110] S. Lee and H. Kim, "GaussianMask: Uncertainty-aware Instance Segmentation based on Gaussian Modeling," in *Proc. 26th International Conference on Pattern Recognition (ICPR 2022)*, pp. 3851-3857, Aug. 2022.

[111] J. Jang, H. Lee, and H. Kim, "Performance Analysis of Phase Change Memory System on Various CNN Inference Workloads," in *Proc. 19th International SoC Design Conference (ISOCC 2022)*, pp. 133-134, Oct. 2022.

[112] J. Jang, H. Lee, and H. Kim, "Characterizing Memory Access Patterns of Various Convolutional Neural Networks for Utilizing Processing-In-Memory," in *Proc. 2023 Int. Conf. on Electronics, Information, and Communications (ICEIC 2023)*, pp. 358-360, Feb. 2023.

[113] D. Chun, J. Choi, H.-J. Lee, and H. Kim, "CP-

CNN: Computational Parallelization of CNN-based Object Detectors in Heterogeneous Embedded Systems for Autonomous Driving," *IEEE Access*, vol. 11, pp. 52812-52823, 2023.

[114] J. Lee and H. Kim, "Discrete Cosine Transformed Images Are Easy To Recognize in Vision Transformers," *IEIE Transactions on Smart Processing & Computing*, vol. 12, no. 1, pp. 48-54, Feb. 2023.

[115] D. Nguyen, N. Hung, H. Kim, and H.-J. Lee, "An Approximate Memory Architecture for Energy Saving in Deep Learning Applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 5, pp. 1588-1601, May 2020.

[116] Mutlu, Onur, et al. "A Modern Primer on Processing in Memory", pp171-243, 2022

[117] S. Lee, H. Lee, H.-J. Lee, and H. Kim, "Evaluation of Various Workloads in Filebench Suitable for Phase-Change Memory," *IEIE Transactions on Smart Processing & Computing*, vol. 10, no. 2, pp. 160-166, Apr. 2021.

[118] S. Cho, "Volatile and Nonvolatile Memory Devices for Neuromorphic and Processing-in-memory Applications," *Journal of Semiconductor Technology and Science*, vol. 22, no. 1, pp.30-46, Feb. 2022.

[119] W. Shim, "Impact of 3D NAND Current Variation on Inference Accuracy for In-memory Computing," *Journal of Semiconductor Technology and Science*, vol. 22, no. 5, pp. 341-345, Oct. 2022.

Ji-Hoon Jang received a BSc and an MSc in electrical and information engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2021 and 2023. Currently, he is conducting research at the same university to obtain a PhD. His current research interests include the areas of the areas of efficient hardware accelerator design for deep neural networks, data prefetcher, phase-change memory system and processing-in-memory architectures.

Jin Shin received an AS in computer system and engineering from the Inha Technical College, Incheon, South Korea, in 2017, a BSc in computer science and engineering from the National Institute for Lifelong Education, Seoul, South Korea, in 2019, and an MSc in Electrical and Information Engineering from Seoul National University of Science and Technology, Seoul, South Korea, in 2021. Currently, he is conducting research at the same university to obtain a PhD. His current research interests include compression and restoration schemes for multimedia applications.

Jun-Tae Park received a BSc and an MSc in Electrical and Information Engineering from the Seoul National University of Science and Technology, Seoul, South Korea, in 2020 and 2023. His current research interests include the areas of efficient hardware accelerator design for deep neural networks and processing-in-memory architectures.

In-Seong Hwang received a BSc in electrical and information Engineering from Seoul National University of Science and Technology, Seoul, South Korea, in 2023. Currently, he is conducting research at the same university to obtain an MSc. His current research interests include the areas of efficient hardware accelerator design for deep neural networks and the design of Processing-In-Memory simulators and controllers.

Hyun Kim received a BSc, an MSc, and a PhD in electrical engineering and computer science from Seoul National University, Seoul, South Korea, in 2009, 2011, and 2015, respectively. From 2015 to 2018, he was a BK Assistant Professor for BK21 Creative Research Engineer Development for IT, Seoul National University. In 2018, he joined the Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, where he is an Associate Professor. His current research interests include algorithm, computer architecture, memory, and SoC design for low-complexity multimedia applications and deep neural networks.