CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Processing-in-Memory based CNN Acceleration: Application Characterization and Simulation

Master's thesis in Embedded Electronic System Design

Lexuan Tang

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Processing-in-Memory based CNN Acceleration: Application Characterization and Simulation

Lexuan Tang

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Processing-in-Memory based CNN Acceleration: Application Characterization and Simulation
Lexuan Tang

Supervisor: Pedro Petersen Moura Trancoso, Computer Science and Engineering
Co-supervisor: Xu Wang, Computer Science and Engineering
Examiner: Per Larsson-Edefors, Microtechnology and Nanoscience

Master's Thesis 2024
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2024

Processing-in-Memory based CNN Acceleration: Application Characterization and Simulation
Lexuan Tang
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Convolutional neural networks (CNN) are widely used in different machine learning tasks, especially computer vision. Large amounts of computation in CNN cause intensive data movement, which makes traditional compute-centric architectures (CPU and GPU) less capable. There is a demand for new memory-centric architecture, such as processing-in-memory (PIM).

Two subcategories of PIM, processing near memory (PNM) and processing using memory (PUM) are discussed in this report. Starting from the application level, we characterize the layers. Then we use the PNM simulator, ramulator-pim, to simulate CPU and PNM architecture and use the PUM simulator, DNN+NeuroSim, to simulate PUM architecture. The CNN models we use are VGG8, UNet, ResNet18 and MobileNetV3. Based on the characteristics of software and advantages of hardware, we aim to accelerate CNN with PIM technologies and estimate the performance improvement by using PIM technologies.

By observing the behavior of different layers on CPU and PNM architecture, we find the correlation between application characteristics and the performance improvement we can get from PNM. The characteristics are memory footprints, number of floating point operations and arithmetic intensity. PUM is another promising technology that provides high speedup compared to CPU with a cost of energy.

Keywords: Processing-in-memory, processing near memory, processing using memory, CNN accelerator

Acknowledgements

I acknowledge the support of our supervisor Pedro Petersen Moura Trancoso, co-supervisor Xu Wang and examiner Per Larsson-Edefors. Their expert guide and insightful feedback is which greatly enhanced the quality of this thesis. I would like to thank Mohammad Ali Maleki and Mateo Vázquez Maceiras for their technical support and constructive suggestions. We are thankful to Chenhao Yang and Jinsheng Bian for giving constructive suggestions on the report. Finally, I would like to thank Daniel Eliasson for the encouragement and support during the thesis.

Lexuan Tang, Gothenburg, July 2024

Contents

1

Introduction

Machine learning models have become an important part of our daily lives. We can find them in products including intelligent robots [1, 2], recommendation systems [3, 4], automotive assistant systems [5, 6], etc. Neural networks are one of the commonly used machine learning models. The first neural network model was first proposed in 1958 as a prototype of a simple two-layer network [7]. Modern neural networks, especially convolutional neural networks (CNNs), become more complex. Complexity brings not only higher accuracy and more capability but also large amounts of parameters and computations [8]. The demand for compute resources is beyond those that traditional central processing units (CPUs) can provide [9]. This demand advances the development of another type of hardware, namely graphic processing units (GPUs).

As specialized hardware for accelerating image processing, GPUs provide more compute resources and architectures optimized for higher parallelism and higher throughput than CPU [10]. These features give GPUs advantages in complex applications with high computation demand, such as neural networks [11]. In 2011, GPUs were first used to accelerate neural network training in ImageNet Large Scale Visual Recognition Challenge [12]. The training time was approximately 5 times faster than on a quad-core CPU [13]. In 2012, an efficient GPU implementation to accelerate convolution, the most important operation in neural networks, was proposed [14]. To exploit the potential of GPUs, programming language (CUDA [15]) and software libraries (cuDNN [16] and RAPIDS [17]) were developed for GPU-based machine learning acceleration [10]. Although GPUs have been proven to be powerful, limited memory bandwidth and intensive data movement restrict their development as machine learning accelerators [18]. While the performance of computation units grows approximately 45% per year, the increase in memory bandwidth cannot keep up with this trend [19]. Due to this gap between computation units and memory, the computation units become underutilized. This causes a decrease in throughput [20]. To do a computation, the data must be brought from main memory, through levels of caches, to register files of computation units. This results in high cost in latency, energy and memory bandwidth [18]. In 2013, an experiment proved that the energy cost of data movement accounts for 18% to 40% of the total dynamic energy, depending on applications [21]. In 2014, another experiment showed that mobile applications spend 34.6% of dynamic energy on data movement on average [22].

Under this circumstance, processing-in-memory (PIM) [18] has gained the attention of the researchers. PIM aims to shorten the distance between computation units

and memory units by enabling computation near memory array or using memory cells [23]. In this way, the latency and energy to move data from memory to processors in traditional systems can be alleviated or even eliminated [18]. There have been decades between PIM being proposed as a concept and its implementation as a commercial product. The development of memory technologies such as 3D-stack memory (HMC [24] and HBM [25]) and non-volatile memory (NVM), extend the possibility of PIM. Processing near memory (PNM) and processing using memory (PUM) are two approaches to implement PIM. There are existing commercial products based on PNM [26, 27, 28]. Some of them are specially designed hardware for machine learning acceleration. In addition, CNN accelerator based on PUM is a popular research topic [29, 30].

1.1 Aim and goals

From all the machine learning models, we choose convolutional neural network (CNN) as our object of research for three reasons: (1) Characteristics of convolution (2) Clear structure and (3) Existing theory support.

First, CNN contains large amounts of convolutions. Substantial computation and memory operations in convolutions consume both time and energy [31]. Thus, they are ideal objects for PIM acceleration. Secondly, CNN models have a clear layer structure so it is easy to observe and analyze the behavior of each layer. Finally, research has shown the feasibility of PIM-based CNN acceleration [32].

To accelerate applications efficiently, software and hardware co-design is necessary. In this report, we aim to understand how to apply PIM technologies, PNM and PUM, to CNN acceleration according to the characteristics of the CNN models. All the experiments are based on simulators. The more specific goals are listed below:

- Characterize the CNN models and the layers in the models.
- Identify the benefits and challenges of using PNM for CNN acceleration.
- Identify the benefits and challenges of using PUM for CNN acceleration.
- Find a method to run the model across different hardware to improve the performance (latency).
- Use simulators to evaluate performance improvement.

1.2 Research questions

By completing the goals above, we would like to answer

- **Question 1** What characteristics make applications more suitable for PNM and PUM?
- **Question 2** What are the advantages and disadvantages of PNM and PUM?
- **Question 3** How can we apply PNM and PUM processing to accelerate CNN?

1.3 Scope and limitation

Our study can provide a high-level understanding of PIM technologies. We focus on the simulation of PIM architectures. The implementation is out of our scope. There are deviations between simulated results and data measured from real hardware. Besides, the accuracy depends on the simulators. The results can vary even if we simulate the same architecture with other simulators. Thus, the conclusions we get in our report are not definite. But we still provide insights into PIM technologies and a simulation flow for future work.

1.4 Organization of the report

The report include five chapters: Introduction, Theory, Method, Results and Conclusion.

In the Introduction chapter, we have motivated our study and listed the research questions we want to explore.

In the Theory chapter, we will explain the concept of CNN and review existing CNN accelerators. Then we will describe typical paradigms of PNM and PUM implementations, show examples of PIM-based CNN accelerators and list common PIM simulators.

In the Methods chapter, we will introduce the simulator tools, simulation flows and the metrics we use for application characterization and performance evaluation.

In the Results chapter, we will present the simulation results and provide the estimated performance improvement by using PIM technologies.

In the conclusion chapter, we will summarize our findings and achievements during the study and answer the research questions.

2

Theory

In this chapter, we will first introduce the concepts of convolutional neural network (CNN) and summarize the advantages and disadvantages of existing CNN accelerators. Then we will explain processing near memory (PNM) and processing using memory (PUM) in detail and present examples of CNN accelerators based on these technologies. Finally, we will list potential choices of PIM simulators.

2.1 Convolutional neural network (CNN)

As shown in figure 2.1, CNN is a subdomain of machine learning.

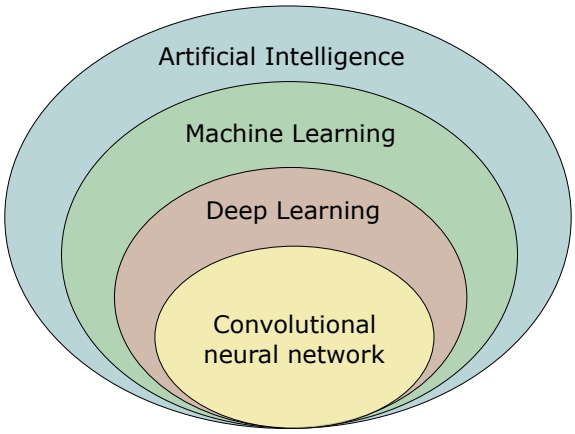


Figure 2.1: The domain of convolution neural network

Machine learning models are compute models with parameters. When machine learning models learn from input data, the parameters are determined to maximize the accuracy of the output. As shown in figure 2.2, this process is called training. The process of a trained model computing output based on input data is called inference. The output can be a prediction, a classification or decisions.

A neural network is a typical machine learning model. It imitates the process of neurons perceiving the environment and passing the signal to connected neurons [33]. Since the input from the environment is not a single value, multiple neurons are needed [7]. These neurons form a layer as shown in figure 2.3. One layer takes input from the previous layer, computes the output and passes the output to the next layer. The number of layers determines the depth of the network and the number of neurons in a layer determines the width.

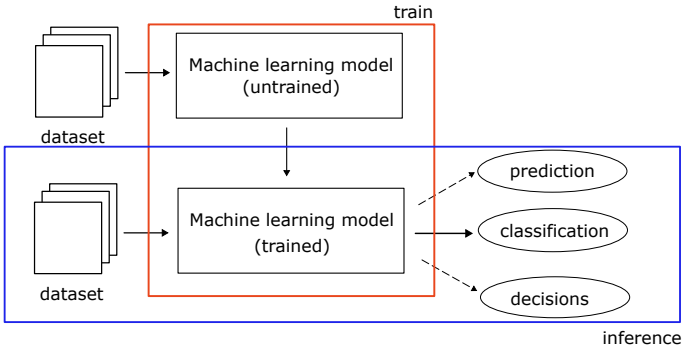


Figure 2.2: Diagram of machine learning model

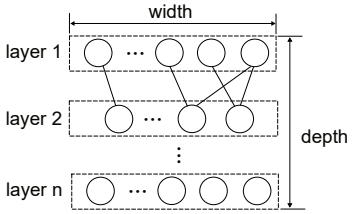


Figure 2.3: Layers in neural networks

As neural networks become deeper and wider [8], a new subdomain of machine learning deep learning based on deep neural networks (DNNs) becomes popular. Typical deep learning models include CNNs [34], recurrent neural networks (RNNs) [35] and transformers [36]. These network models with various structures have advantages in different tasks. CNNs contain multiple convolution layers that are good at extracting features from images. They are widely used in image classification, object detection and image segmentation [8]. RNNs have memory cells storing the previous states. They have advantages in processing sequences, such as speech recognition and natural language translation [37]. Transformers introduce the attention mechanism that allows the model to find dependency between elements in sequence regardless of distance [36]. They have good performance in both natural language processing and computer vision [38].

CNN is a subdomain of deep learning that is good at processing images. CNNs can also be applied to other input data. Depending on the dimension of input data, CNN can be categorized as one-dimensional (1-D) CNN, two-dimensional (2-D) CNN and three-dimensional (3-D) CNN [8]. 1-D CNNs can be applied to natural language processing [39]. 2-D CNNs are the mainstream model for computer vision [40, 41] and image processing [42, 43]. 3-D CNN can be used for video processing [44] or 3-D image processing [45]. These CNN models have similar layer structures.

Convolution layers

Convolution layers are the most important layers in CNN models. The convolution operation is similar to the filtering that can extract the features including high-frequency components (edge) or low-frequency components (continuous area). A 2-D convolution can be expressed as equation (2.1). (O is output of i*j, F is filter

of n*m, I is input.)

$$O(i,j) = \sum_n \sum_m F(n,m) * I(n+i, m+j) \tag{2.1}$$

The difference between convolution in CNN and 2-D convolution is that the output is the sum of the convolution results of all inputs, as shown in figure 2.4. In this way, the feature can be fully extracted. The parameters in the filter are weights.

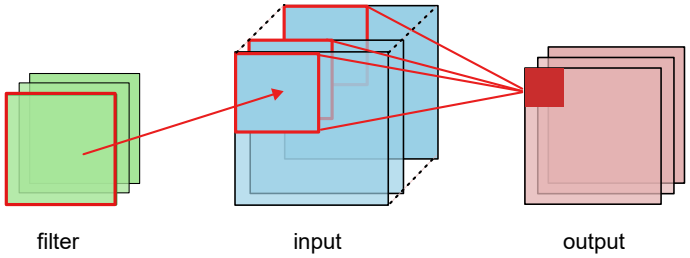
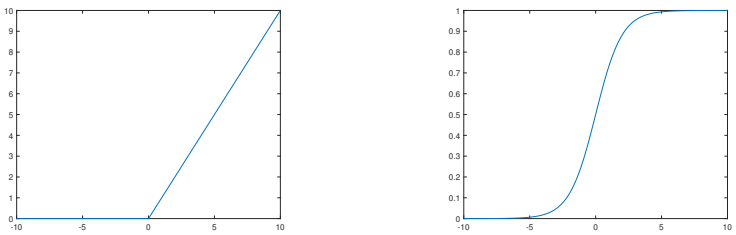


Figure 2.4: Convolution layer

Activation layers, pooling, batch normalization, fully connected layers and blocks

Except for convolution layers, CNN can contain activation layers, pooling layers and fully connected layers. The activation functions following the convolution operations form the activation layers that add non-linearity to the model. Figure 2.5 shows two common activation functions Rectified Linear Unit (ReLU) [46] and Sigmoid function. After the features are extracted by the convolution layers and non-linearity is added by activation layers, pooling layers are used to get higher-level features. The pooling operation is similar to downsampling. Common pooling layers include max pooling and average sampling. Batch normalization layers normalize the input value to enhance the stability of the model. Fully connected layers are usually the final layers in the CNN models. They act as the classifier that converts the features from convolution layers to the output. Another characteristic is that the input and the output for a fully connected layer are all one-dimensional.

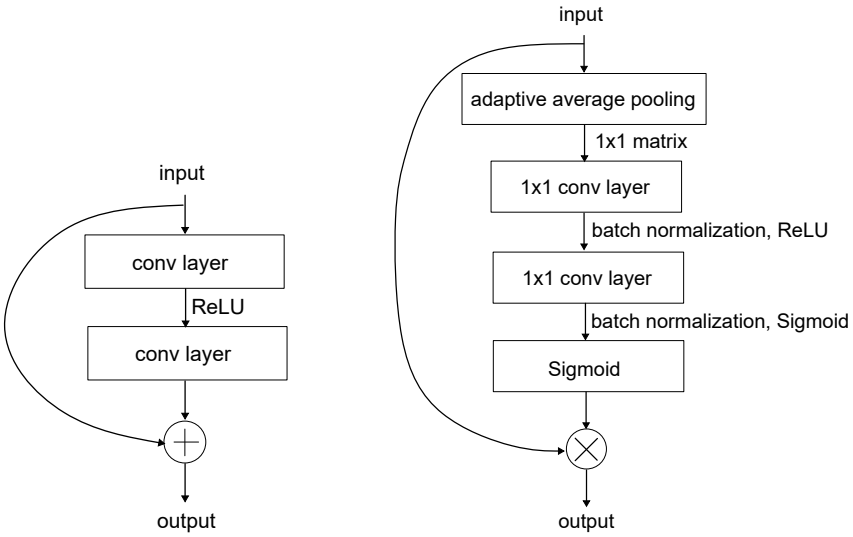


(a) ReLU **(b)** Sigmoid

Figure 2.5: Example of activation functions.

Early CNN models have simple layered structures. As the complexity of the model increases, block becomes a new unit that is larger than layer and smaller than model.

It can improve the effectiveness of model design and training. The block structure is used in both ResNet and MobileNet. Figure 2.6 shows the residual block of ResNet18 [47] and the Squeeze-and-Excitation (SE) block of MobileNetV3 [48].



(a) Residual block as described as in [47]

(b) SE Module as described as in [49]

Figure 2.6: Example of blocks in CNNs.

Table 2.1 lists typical CNNs. LeNet-5 is one of the oldest and simplest CNNs for handwriting recognition [50]. AlexNet is the winner of 2012 ImageNet Large Scale Visual Recognition Challenge. Besides improving image classification accuracy, it also caused attention to the depth of neural network models [51]. As the 2014 ImageNet Large Scale Visual Recognition Challenge winner, VGGNet has a deeper architecture (up to 19 layers) and uses small kernels [52]. Problems such as gradient vanishing and gradient exploding become unavoidable when neural networks become deeper [53]. ResNet introduces the concept of residual learning to solve these problems in conventional deep CNNs [47]. The fast development of CNNs pushed the models forward to different application scenarios including large data centers and small mobile devices. Models such as MobileNet are specially optimized for mobile applications. Benefiting from its lightness and efficiency, it can be embedded with mobile and complete tasks including objection detection, face recognition, large scale geolocalization, etc [54].

2.2 CNN accelerator

The acceleration of convolution is the most important part of the acceleration of CNN. Approximately 90% of the operations in models are contributed by convolution layers [55]. To avoid the data movement problem, new dataflows, such as output stationary, weight stationary and row stationary, are proposed. ShiDianNao uses an output stationary dataflow that allows computation units to communicate with each

Table 2.1: Typical CNNs

CNN	# Layers(Convolution layers)	# Parameters
LeNet-5 [50]	7(2)	6M
AlexNet [51]	8 (5)	60M
VGGNet8 [50]	12 (6)	13M
ResNet18 [47]	20 (16)	11M
MobileNetV1 [54]	30 (13)	4M

other [56]. In this way, the output of one computation unit can be directly fetched through registers by other units. Weight stationary dataflow used by nn-X [57] and row stationery used by Eyeriss [58] are similar. The nn-X reuses the weights and Eyeriss reuses parameters including weights, inputs and temporary outputs. Although the on-chip data movement can be alleviated by well-designed dataflow, the data movement from off-chip memory to on-chip buffer still makes the design less efficient [58].

Besides dataflow, the hardware platform is another design choice. Besides CPUs and GPUs, field programmable gate array (FPGA) and application-specific integrated circuit (ASIC) are other two choices. Accelerators based on field programmable gate array (FPGA) can get higher speed with less energy consumption than CPUs and GPUs [59]. They have been applied to CNN acceleration [60] and RNN acceleration [61]. But the limited computation resources (DSPs) are the challenge for FPGA-based accelerators [59]. ASIC-based accelerators include energy-efficiency research chips such as Eyeriss to high-performance chips in data centers such as TPU [62]. But the optimization of data flow to the design of computation units still doesn't change the fact that all CPUs, CPUs, FPGAs and ASICs centralize the computation units instead of the memory. Thus they still suffer from intensive data movement and limited memory bandwidth [63]. Under this circumstance, PIM is seen as a powerful candidate for future accelerators.

2.3 Processing-in-memory (PIM)

Under the circumstance that the existing accelerators suffer memory bottleneck or data movement, more attention is turned to PIM.

The earliest prototype of PIM, logic-in-memory, was proposed in 1970 [64]. It suggested that intelligent caches that can do comparison and addition operations can save costs in both computation and data movement. Due to the limitations of manufacture and technologies, PIM stayed in the theoretical stage. With the support of new memory technologies such as non-volatile memory, PIM has been applied to domains, including graph processing [65, 66, 67], databases [68, 69, 70], and machine learning [71, 72, 73].

2.3.1 Processing near memory (PNM)

The common paradigm of PNM is integrating processing units in different types of memory [74] including traditional double data rate synchronous dynamic random access memory (DDR) and emerging 3D stack memory. The processing units are usually integrated at the bank level. The memory banks in DDR and 3D stack memory consist of many memory cells as shown in figure 2.7.

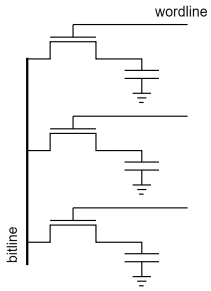


Figure 2.7: DRAM cells

New technologies including through silicon via (TSV) and die stacking technology advance the development of 3D stacked memory.

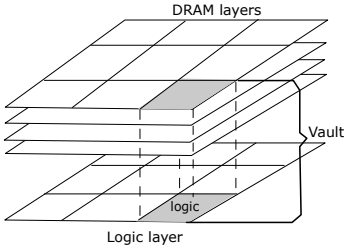


Figure 2.8: The structure of HMC as described in [24]

Hybrid memory cube (HMC) [24] and stacked dies based on high bandwidth memory (HBM) standard [75, 76] are produced and tested. The structure of HMC is shown in figure 2.8. A HMC stack can be divided into several vaults. A vault contains some logic including a memory controller and refresh management, and memory arrays connected together with TSVs. The processing elements can be integrated into the logic layer. This is a great progress for implementing PNM.

There are examples of PNM products and their application on machine learning acceleration: UPMEM announced its first commercial PNM system based on DDR4 in 2021 [26]. It integrates a simple processor, DRAM processing unit (DPU), into the DDR memory arrays. Figure 2.9 shows the architecture of PIM-enable memory, PIM chip. One PIM chip contains the 8 DPUs with instruction and working RAM (similar to instruction and data caches), a direct memory access (DMA) engine and a 64MB DRAM bank. It is proven that this hardware can accelerate sparse matrix-multiplication [77] and weightless neural network (WNN) [78].

In the same year, Samsung proposed their FiMDRAM architecture, which adds a single-instruction-multi-data (SIMD) processor into HBM2 bank [28], as shown in

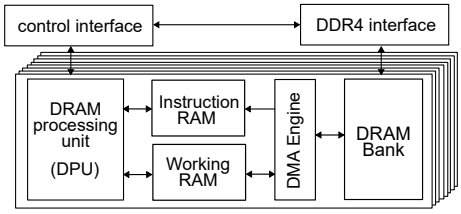


Figure 2.9: The architecture of UPMEM PNM chip as described in [26]

figure 2.10. After adding the processors, data buses are also added to the memory chip to handle the communication of the processors.

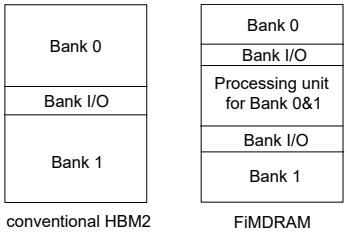


Figure 2.10: FiMDRAM banks contrasted with conventional HBM2 banks as described in [28]

They implemented the FiMDRAM as a coprocessor to a GPU and ran DeepSpeech2 [79] on the system. Compared to the GPU system, the hybrid system gets a speedup of 2.1 times and saves 71% of system energy. In 2022 SK Hynix proposed another architecture, Aim, that adds processor units with mainly multiplier, adder and accumulator to GDDR6 memory [27]. This architecture can improve the performance of RNN by approximately eight times and the performance of long-short-term memory (LSTM) [80] by approximately ten times. These implementations prove the feasibility of PNM, but there are still problems to be solved. First, the characteristics of applications suitable for PNM need to be identified [18]. Second, the thermal and power constraints for 3D stack memory are different from normal memory. Although it is possible to integrate a general processor in the logic layer of 3D stack memory, the power needs to be considered [81].

2.3.2 Processing using memory (PUM)

The principle of PUM is to take advantage of the memory cell and minimize the modification on it to add computation ability to the circuit [18]. DRAM is the mainstream choice for main memory now. RowClone enables bulk data operations including copy and initialization in memory by modifying the row activation behavior of DRAM [82]. Ambit [83] uses charge sharing among cells connected to the same bit line to implement bitwise operations including AND, OR, NOT and combinations of them. Compared to a processor, PUM based on DRAM can only handle very simple operations. Besides, the development of DRAM has reached capacity bottlenecks, high-cost interconnects and power dissipation on regular refresh [84].

Emerging non-volatile memory (NVM), such as phase change memory (PCM) [85] and resistive random-access memory (ReRAM) [86], that can be used for PUM were then proposed. NVM-based accelerators perform matrix multiplication inside the memory array. A ReRAM array is shown in figure 2.11. PRIME implements a neural network accelerator that can compute matrix multiplication and pooling inside ReRAM array [29]. PipeLayer is an accelerator that supports both inference and training of neural networks and adds intra-layer parallelism. The average speedup is approximately 40 times compared to CPU [87]. An accelerator based on PCM is also proposed recently and tested with LSTM benchmark [88]. There are still challenges

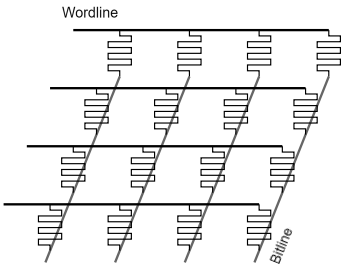


Figure 2.11: The ReRAM array as described in [29]

for PUM: First, the operations that can be done using memory cells are limited. In PRIME, except for matrix multiplication other operations such as pooling and activation functions are performed by extra circuits or host processors [29]. Second, the accuracy of computing using memory cells is an important consideration when applying PUM to CNN acceleration [55]. Third, PUM can save the energy of data movement but it will bring other energy costs. One example is the activation of multiple rows and columns [89].

2.3.3 Simulators

Simulators are the mainstream tools in research on PIM [90]. By analyzing simulation results, we can identify the benefits and limitations of PIM architectures, identify the application suitable for PIM and assessing the benefits of using PIM [18].

Table 2.2 summarizes the existing simulators for PNM and PUM technologies.

Table 2.2: Existing simulators

Simulator	Type	Supported memory
ramulator-pim [91]	PNM	DRAM, HBM, HMC
PIMSimulator [92]	PNM	HBM
MultiPIM [93]	PNM	HMC
DNN+NeuroSim [94]	PUM	SRAM, NVM

Ramulator-pim [91] is a PNM simulator that contains a processor simulator and memory simulator. PIMSimulator [92] simulates single-instruction multi-data cores that support operations including addition, multiplication, multiply-accumulate,

etc. MultiPIM [93] is a simulator based on ramulator-pim with additional support for virtual memory and coherence management. DNN+NeuroSim [94] is a PUM simulator that provides flexibility at both the application level and hardware implementation level.

3

Methods

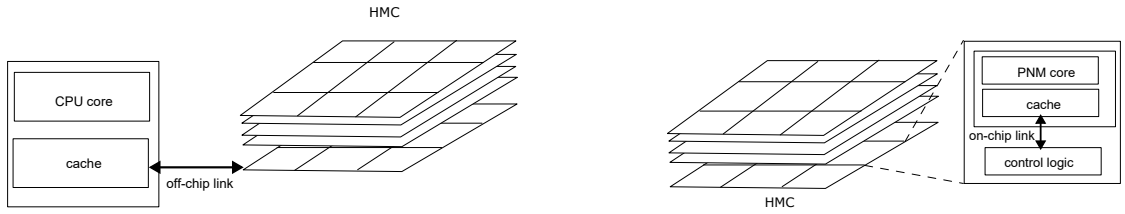
In this chapter, we will first introduce two simulators we use, ramulator-pim and DNN+NeuroSim. Then we will describe the working flow to use these simulators to get the latency and energy data. The metrics for data analysis will also be discussed. Finally, we show the process of performance improvement estimattion.

3.1 Simulator tools

As discussed in section 2.3.3, the possible choices for PNM simulators are ramulator-pim, PIMSimulator and MultiPIM. The application supported by PIMSimulator is limited. MultiPIM is an extension of ramulator-pim but the virtual memory and coherence management are unnecessary for us. Thus, we choose ramulator-pim as our PNM simulator. DNN+NeuroSim is the PUM we found that matches our needs.

3.1.1 Ramulator-pim

Ramulator-pim is a PNM simulator that takes an executable file as input and provides an instrumentation interface that can choose a specific part of the application as offload. It is possible to combine ramulator-pim with the energy simulator, DRAMPower [95].



(a) CPU core **(b)** memory core

Figure 3.1: The simulation setup of CPU core and memory core.

Ramulator-pim contains two cooperating simulators: ZSim and ramulator. In our project, we use ZSim to simulate a processor running the compiled neural network models. During the run time, the memory access requests are recorded and dumped into a trace file. Then ramulator takes the trace file as input and simulates the memory reading and writing operations. The latency of the CPU system contains

three parts: the cache miss latency (from ZSim), the memory read latency (from ramulator) and the computation time of the CPU (from ramulator). The latency for the PNM simulation can be directly provided by ramulator. The architectures of the CPU core and the PNM core are shown in figure 3.1. The most significant difference between them is that the PNM core is located in the logic layer in HMC. The latency and energy spent on off-chip traffic are avoided.

3.1.2 DNN+NeuroSim

DNN+NeuroSim [94] is a PUM simulator that provides flexibility at both the application level and hardware implementation level. It contains a performance evaluation tool, NeuroSim, wrapped with a Pytorch wrapper that is compatible with different neural network models. As shown in figure 3.2, we first run model inference in Pytorch framework, capture the input and weight data in each layer, and store them as .csv files. Then we use these files as input to NeuroSim to get detailed results on the hardware cost, latency and energy consumption on each layer.

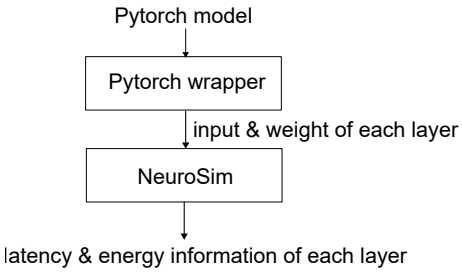


Figure 3.2: Working flow of DNN+NeuroSim

The NeuroSim can generate an optimized hierarchical PUM circuit and executes the inference on the simulated circuit. As shown in figure 3.3, from the top down, the chips consist of tiles, processing elements and synaptic arrays. Synaptic arrays are the basic units of PUM.

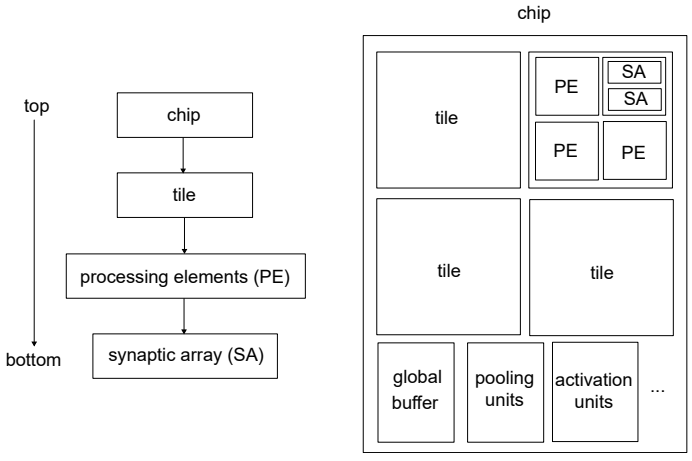


Figure 3.3: Hierarchy of DNN+NeuroSim

As shown in figure 3.4, the synaptic array contains a memory array and peripheral circuits managing the reading and writing of the memory array. When generating the circuits, the simulator reads the layer information of the models from a prepared csv file, maps the weights into the memory. The synaptic array aims to accelerate the matrix multiplication. Besides convolution layers, other layers including pooling layers, activation layers, batch normalization layers are all implemented as additional circuits in the chip.

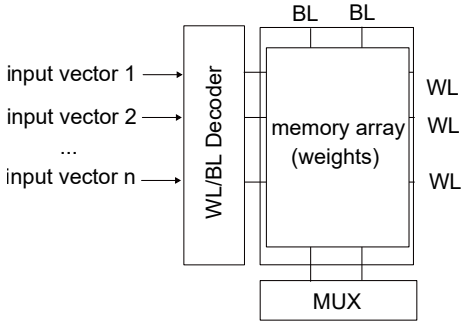


Figure 3.4: A synaptic array in DNN+NeuroSim (Sequential mode) as described in [94]

The inference process is described in figure 3.4 This PUM architecture aims to accelerate the convolution layers, or matrix multiplication. There are two steps in matrix multiplication: multiplication and accumulation. The weights are saved in the memory array before inference. The input matrix is divided into vectors and sent into the memory array. The multiplication is done by memory cells (SRAM cell or ReRAM cell). If the multiplication results are read out row by row and accumulated by an additional circuit outside the memory array, the simulator is in sequential mode. If the multiplication results are accumulated on the bit lines (BLs), the simulator is in parallel mode. The results collected from the BLs are analog, for future computation and communication, an analog-to-digital converter (ADC) is needed. The precision of this ADC affects latency, energy and also area. The technologies supported by DNN+NeuroSim are listed in table 3.1.

Table 3.1: Supported technologies in DNN+NeuroSim

Memory cell type	Technologies nodes (nm)
SRAM	7,10,14,22,32,45,65,90,130
ReRAM	22, 90, 130
FeFET	22

3.2 Simulation flow

Our project is based on simulation. By using simulators, we aim to identify the benefits and the challenges of PIM technologies and measure performance improvement. The simulation flow is shown in figure 3.5.

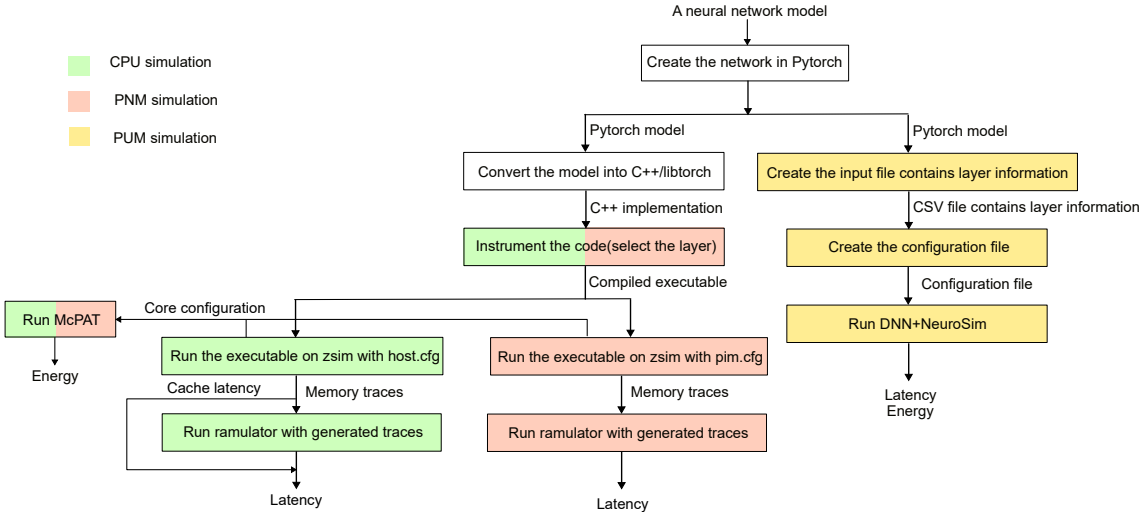


Figure 3.5: Simulation flow

There are three types of simulations: CPU simulation, PNM simulation and PUM simulation. We implement a neural network model in Pytorch framework using Python. Based on the requirements of the simulators, the Pytorch model can be directed used as input to the DNN+NeuroSim. The latency and energy information can be directly fetched from the output file of DNN+NeuroSim. For CPU and PNM simulation, The Pytorch model is first converted into a C++/Libtorch implementation. This C++ implementation is compiled into an executable file. The Zsim runs the executable file with different configuration files for CPU and PNM architecture and provides a memory trace file for ramulator. The ramulator simulates the memory behavior based on this file. The latency of CPU simulation contains three parts: the computing time of the processor, the cache miss latency and the memory read latency. The cache miss latency is from ZSim and other two latencies are from ramulator. The latency of PNM simulation is from ramulator and the latency of PUM simulation is from DNN+NeuroSim. Ramulator can provide the energy information for memory but not for the core. We measured the energy and area cost of the CPU core and PNM core by McPAT [96].

3.3 Metrics

The analysis is based on metrics we evaluate in simulation and calculate. The most important metric we use to evaluate performance is latency. Besides, we also measured energy for better comparison across the hardware. The metrics we calculate for application characterization include memory footprint, FLOPS and arithmetic intensity. The method to estimate memory footprint and arithmetic intensity is introduced in the following sections.

Memory footprint

Memory footprint quantitatively describes the memory taken up by the program during run time. PIM is proposed as a promising way to accelerate neural networks

because it can overcome the memory bottleneck and alleviate data movement problems in traditional systems. The memory footprint then becomes an important characteristic that affects the run time performance on the CPU and PIM system. There are existing researches on PIM technologies using memory footprints as a metric [63, 97]. In our project, we use the total amount of inputs and weights participating in computation to approximate the memory footprint. Convolution layers in neural networks are usually computation-intensive, since the same multiply and add operation is done on hundreds of data. The memory used for data is much larger than the memory used for the program itself. Thus the total amount of activations (or inputs) and weights of a layer is used as an approximation to the memory footprint of that layer.

Floating-point operations (FLOPS)

The FLOPS is the total amount of floating-point operations. We use torchsummaryx library to extract this information from the Pytorch models.

Arithmetic intensity

The other metric we use is arithmetic intensity that is calculated as

$$\text{Arithmetic Intensity} = \frac{\text{total amount of FLOPS}}{\text{total amount of parameters}} \tag{3.1}$$

Different from its original definition [98], we ignore the effects of hardware in our project. One example is that the memory bandwidth of a CPU system and a PNM system are not the same. Although the hardware factors are not considered in the equations, they can still provide a reference for the performance of the PIM architectures.

3.4 Application partition

Different layers are suitable for different hardware [63]. First, we characterize the layers by memory footprint, FLOPS and arithmetic intensity metrics. To maximize the performance improvement from using PIM, we need to choose the best hardware for each layer. Here, the best means that the hardware which can provide the shortest latency. According to the understanding of application characteristics and advantages of PIM technologies, we predict the best hardware for the layer. The predictions are validated by running benchmark models on the simulators. Then we partition the CNN models into layers, assign their execution on the best hardware for them and run the layers on the simulator for this best hardware.

```
latency_baseline = 0
latency_PIMaccelerated = 0

for layers in the CNN model
    // Run simulation on CPU simulator and get latency
    latency_baseline += latency
```

```
done

for layers in the CNN model
    // Characterize the layer and find the best hardware
    // Run simulation on
       the simulator for the best hardware and get latency
    latency_PIMaccelerated += latency
done

speedUp = latency_baseline / latency_PIMaccelerated
```

Finally, we will measure the improvement in latency using simulators. The pseudo-code above describes the process to calculate an estimated performance improvement.

4

Results

In this chapter, we will first present the setups and benchmarks for our study. Then we show the results of application characterization and prediction of suitable hardware. We will use the simulation results from PNM simulator to validate our prediction and estimate the performance improvement by using PNM. Finally, we analyze the PUM simulation results and summerize the advantage and disadvantage of simulated PUM architecture.

4.1 Experiment setup

The simulated architectures are presented in section 3.1. The configurations of PNM processing are listed in table 4.1.

Table 4.1: Configurations of PNM simulation

Parameter	CPU core	PNM core
Type	out-of-order	out-of-order
Number of cores	4	4
L1 Cache size	32K(D)+32K(I)	32K
L2 Cache size	256K	-
L3 Cache size	8M	-
Frequency	4GHz	1.25GHz

We use 22nm ReRAM for PUM simulation. Other constraints or setups for the simulations are listed in table 4.2.

Table 4.2: Configurations of PUM simulation

Parameter	Value
Read-out mode	parallel mode
Precision of memory cell	2 bits per cell
Precision of ADC	5 bits

For CNN models, we choose models based on three criteria: availability, simplicity and heterogeneity . Availability is our first consideration. The default models in the simulators or models with open-source implementation are preferable. Simplicity

means that both network and the implementation need to have a clear structure. The heterogeneity in memory footprints, FLOPS and arithmetic intensity can help us to find the correlation between application characteristics and hardware advantages.

Our final choices are VGG8 [52], UNet [99], ResNet18 [47] and MobileNetV3 [49]. VGG8 and ResNet18 are the default models in the PUM simulator. The implementations of UNet and MobileNetV3 models are open-source. These four models have different structures. VGG8 contains convolution layers and fully connected layers. UNet has a symmetry structure constituting convolution layers. ResNet18 and MobileNetV3 are implemented in blocks, containing convolution layers, batch normalization layers and nonlinear layers. Besides, MobileNetV3 contains SE blocks that consist of 1x1 convolutions. Compared with normal convolutions, the amount of computations is significantly reduced [54]. We aim to find how PIM technologies affect the performance of these blocks.

4.2 Application characterization

Before the simulation, we divide all 99 layers in four models into four families as shown in figure 4.1 based on three metrics, memory footprint, FLOPs and arithmetic intensity. The detailed classification criteria are listed in table 4.3.

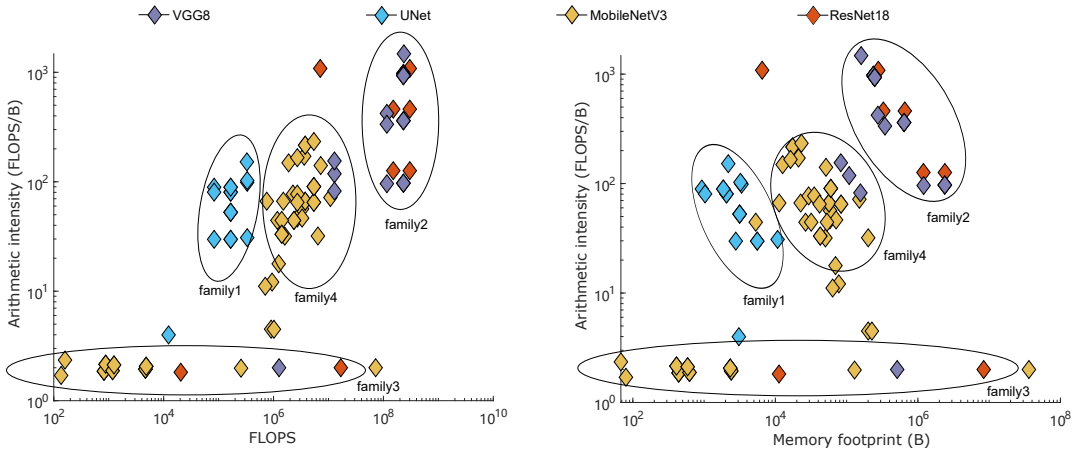


Figure 4.1: Arithmetic intensity vs number of FLOPS (left) and arithmetic intensity vs memory footprint across layers of four models (right)

Table 4.3: Classification of four families

Family	Arithmetic intensity	Memory footprint	FLOPs
family1	10-100	1K-10K	0.1M-1M
family2	100-1000	0.5M-10M	>100M
family3	1-10	100B-10M	100-10M
family4	10-100	10K-0.5M	1M-10M

with the speedup data we get from different layers, we summarize the correlation

between the characteristics of the families and the suitable hardware. By suitable, we mean the hardware that can provide shorter latency.

Table 4.4: Characteristics of four families

Family	Arithmetic intensity	Memory footprint	FLOPS	Suitable hardware
family1	medium	small	medium	CPU/PUM
family2	high	large	large	CPU/PUM
family3	very low	small-large	small-large	PNM/PUM
family4	medium	medium	medium	CPU/PNM/PUM

Family1 and family2 have better performance on CPU. For family1, a small memory footprint limits the advantage of PNM in memory accessing time. Thus it is more suitable for CPU. For family2, the large amount of FLOPS magnify the disadvantage of PNM in computation time. As a result, layers in family2 perform better on CPU. Family3 contains a layer with very low arithmetic intensity. That means the computation is done at same or similar speed as memory access. Under this circumstance, the CPU core suffers from long memory access latency that is avoided by PNM. So, the family3 is more suitable for PNM. Characteristics of layers in family4 are neutral. The choice of hardware depends on the specific information of the layer. Since the PUM simulator provides specific hardware for each model, we believe that every layer can benefit a lot from PUM. The comparison between CPU and PUM or PNM and PUM will be discussed later.

4.3 PNM

In table 4.5, we present the speedup of different models using PNM. The baseline we use is the CPU core with the configuration in table 4.1. All four models have better performance on CPU. To get a better understanding, we observe the behavior of each layer separately.

Table 4.5: Speedup of four models using PNM

Model	Speedup
VGG8	0.51
UNet	0.05
ResNet18	0.24
MobileNetV3	0.56

In figure 4.2, we present the speedup of different families. As predicted, the layers in family3 are more suitable for PNM than those in other families. Layers in family1 and family2 are more suitable for CPU. Based on the high-level understanding of the connection between application characteristics and advantages of hardware, we will analyze every model in detail.

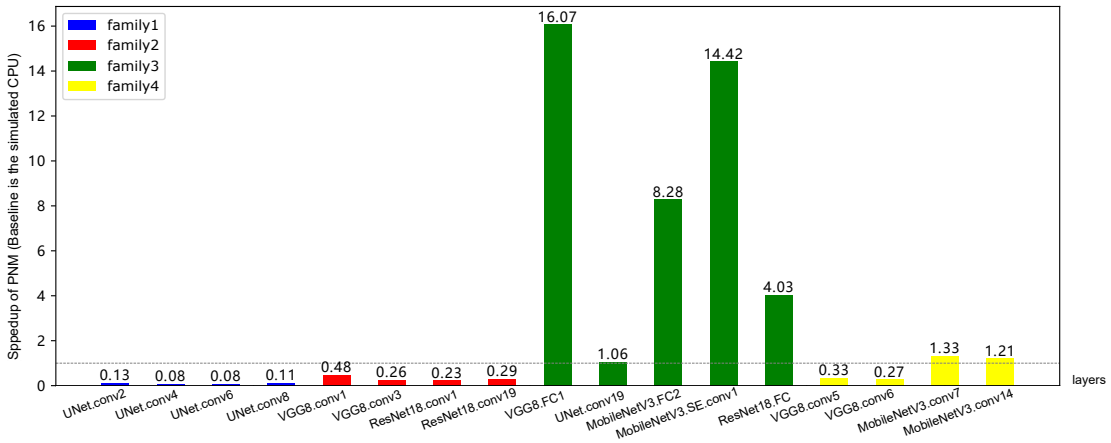


Figure 4.2: Speedup of layers in different families from four families (Baseline is the simulated CPU)

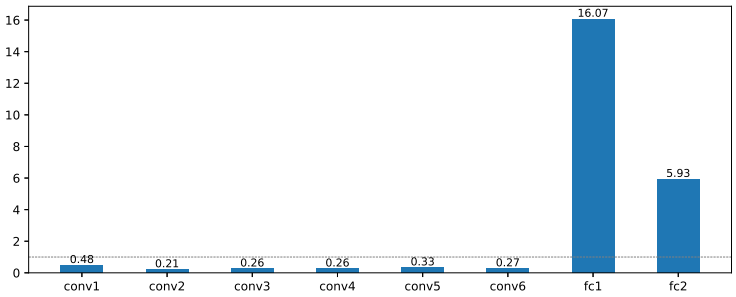


Figure 4.3: Speedup of VGG8 on PNM simulator (Baseline is the simulated CPU)

VGG8

As shown in figure 4.3, there is no performance improvement brought by near-memory processing for any of the six convolution layers. This is in contrast to the two fully connected layers which have different levels of speedup. The structure and the metrics of VGG8 are listed in Appendix A.

The two fully connected layers have lower arithmetic intensity than the convolution layers. The fc1 and fc2 have similar arithmetic intensity but the memory footprint of fc2 is lower than fc1. As mentioned in table 4.1, the frequency of the CPU is approximately 3 times faster than the PNM core. For an application with a small memory footprint, the advantage of PNM in short memory access is weakened by the short computation time of the CPU. Thus, the speedup of fc2 is lower than fc1.

UNet

The structure and the metrics of UNet are listed in Appendix A. Different from VGG8, UNet has a symmetric structure. This symmetry can be observed in figure 4.4. When the memory footprints and the arithmetic intensity of the two layers are the same, the speedup of them are the same. Similar to VGG8, the

conv19 with the lowest arithmetic intensity can benefit from PNM.

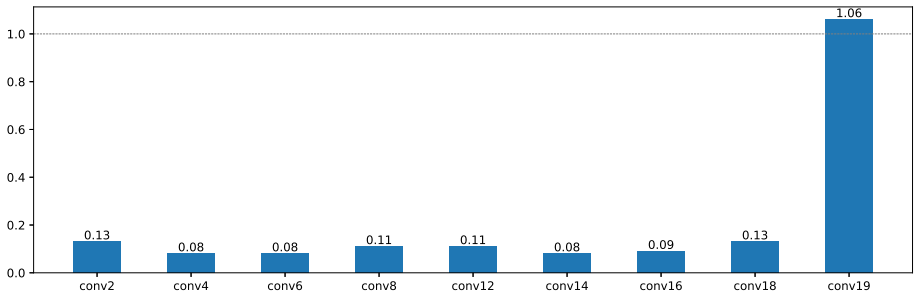


Figure 4.4: Speedup of UNet on PNM simulator
(Baseline is the simulated CPU)

ResNet

For ResNet18, we compare the speedup of normal convolution layers, downsample convolution layers that have 1x1 kernels, and a fully connected layer listed in the appendix. Since the downsample layers have lower arithmetic intensity than conv1 and conv2, they have relatively higher speedup according to figure 4.5. But compared with the fully connected layer, they still count as compute-intensive in Appendix A. And similar to VGG8 and UNet, the fully connected layer still benefits from PNM.

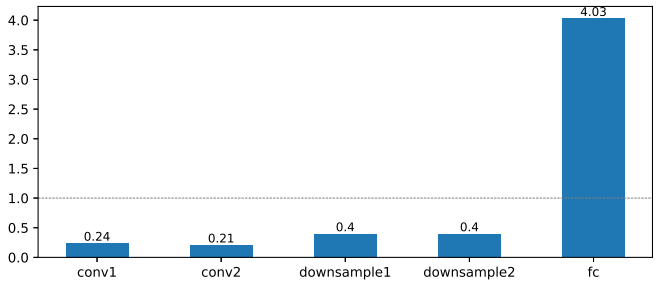


Figure 4.5: Speedup of ResNet18 on PNM simulator
(Baseline is the simulated CPU)

MobileNet

For MobileNetV3 we compare different convolution layers and fully connected layers together. Similar to the VGG8, UNet and ResNet, normal convolution layers, conv1, conv2 and conv3 are more suitable for CPU execution. Different from ResNet, the speedup of two convolution layers in SE modules with 1x1 kernels are approximately 10x. As listed in Appendix A, SE.conv1, SE.conv2 and two fully connected layer fc2 all have vert low arithmetic intensity. Thus, they are suitable for PNM. This observation is consist with before, but the fc1 have better performance on CPU.

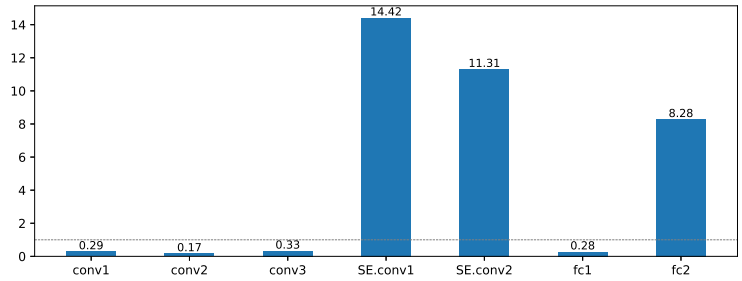


Figure 4.6: Speedup of MobileNetV3 on PNM simulator (Baseline is the simulated CPU)

As shown in table 4.6, for fc2 the difference between the computation time of near-memory core and CPU core is covered by the memory access latency in CPU core. For fc1, the advantage of PNM is weakened by the long computation time of PNM core.

Table 4.6: CPU execution time breakdown of fc1 and fc2

Layer	CPU computation time (µs)	CPU memory access time (µs)
fc1	2380	1940
fc2	18	876
Layer	PNM core computation time (µs)	
fc1	14200	
fc2	87	

Estimated performance improvement

Based on the results above, we partition the models into layers and assign them to the suitable hardware. The estimated performance improvement from the simulator is shown in table 4.7. Compared with VGG8 and MobileNetV3, UNet and ResNet18 have quite low speedup. This is because there is only one layer that can be accelerated by PNM and these layers account for a small portion of the total execution time of the models. These data are from ramulator-pim simulator using a specific architecture. Instead of proving how much performance improvement we can get from using PNM, we want to show that it is possible to accelerate CNN with PNM and a rough estimation of speedup.

Energy Analysis

After the discussion on latency, we explore the energy improvement brought by PNM. As shown in table 4.8, PNM can reduce the energy consumption for most layers. Since the PNM architecture we simulate has a core slower than CPU and nearer to the memory, the energy spent on computation and memory access can be reduced.

Table 4.7: Speedup of four models using PNM

Model	CPU (ms)	CPU+PNM (ms)	Speedup
VGG8	177.99	91.96	1.94
UNet	42.94	42.93	1.0002
ResNet18	288.96	288.86	1.0003
MobileNetV3	1405.51	1104.44	1.27

Table 4.8: Estimated energy improvement of layers in VGG8 using PNM

Layer	Improvement in energy consumption
conv1	1.12x
conv2	0.91x
conv3	1.11x
conv4	1.13x
conv5	1.26x
conv6	1.23x
fc1	43.01x
fc2	7.55x

4.4 PUM

The PUM simulator we use generates specialized hardware to accelerate the inference of the network models. For every model, there is a specific designed circuit. Since the PNM processing and the CPU have similar architectures, it is reasonable to compare them side by side. But PUM simulator provides different technology and architecture. Thus in this section, we will compare the PUM and CPU and present data to show the advantages and disadvantages of PUM.

In table 4.9, we listed the speed of four models using PUM. The baseline is still the CPU core with the configuration in table 4.1.

Table 4.9: Speedup of four models using PUM

Model	Speedup
VGG8	70.44
UNet	6.93
ResNet18	21.34
MobileNetV3	157.11

PUM can provide a great performance improvement. And since the hardware is optimized according to the application, the characteristics of layers are weakened. One example is that the convolution operations are mapped into multiplication in memory cells. One input is involved in one computation. Thus, the arithmetic intensity is a constant number. However, the memory footprint is still an important

characteristic. Since the more parameters involved, the more memory cells are used. To validate our prediction, we still break the models down into layers.

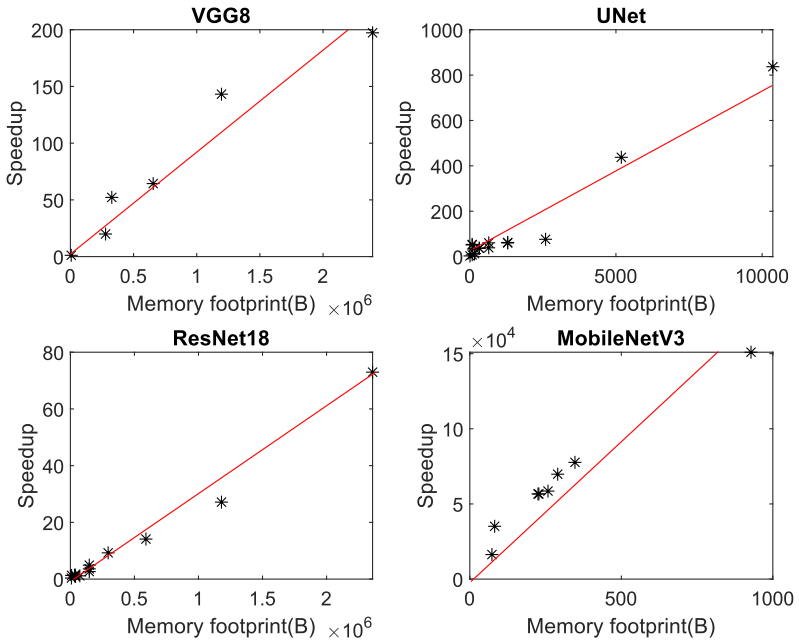


Figure 4.7: Speedup vs memory footprint of layers in four models (Baseline is the simulated CPU)

As shown in figure 4.7, we can see that the larger the memory footprint is, the more speedup we can get from using PUM. This is a reasonable result. Compared to PUM, CPU needs time to fetch the parameters for computation. The larger the memory footprint is, the longer the memory access time for CPU is. Since the PUM saves the memory access time, it can gain more speedup when more parameters are involved. But we also observe an increase in energy as the memory footprint increases. For PUM the parameters for computation need to be stored in memory cells. The larger the memory footprint is, the more memory cells are used. As a result, the dynamic energy increases.

Besides the trade-off between energy consumption and speedup, we also find some impractical aspects of this tool. One example is that the scalability with input size of this architecture is not good. We run the VGG8 and UNet on DNN+NeuroSim with 32x32 and 224x224 inputs. The results are shown in table 4.10. For both VGG8 and UNet, the increase in input size directly leads to the area cost, latency and energy consumption.

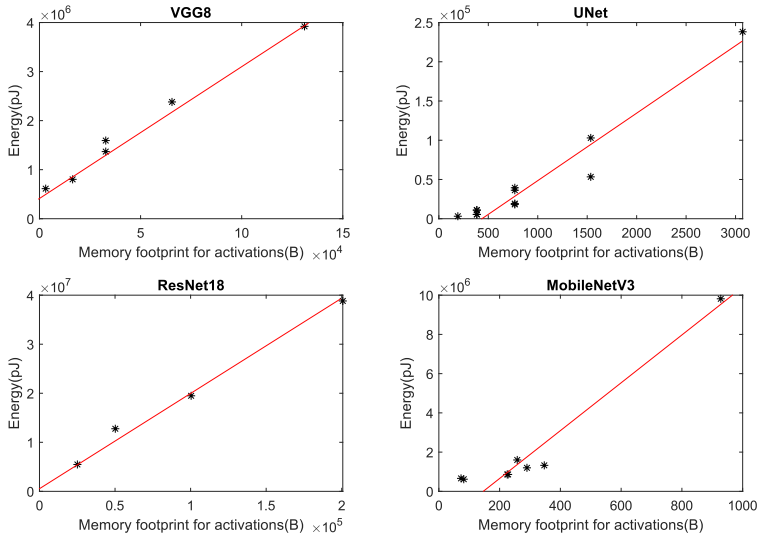


Figure 4.8: Dynamic energy vs memory footprint of layers in four models (Baseline is the simulated CPU)

Table 4.10: Area, latency and energy comparison between different input size

Model	Input size	Area (mm^2)	Latency (ms)	Dynamic energy (nJ)	Leakage energy (nJ)
VGG8	32x32	58.7	1.36	50.72	1.48
	224x224	1311.0	934.95	6919.69	1760
UNet	32x32	3.2	0.58	80.58	0.03
	224x224	18.8	136.44	363.92	6.92

5

Conclusion

In our report, we used a simulated CPU as our baseline and estimated the performance improvement we can get from using PNM and PUM with simulators.

The characteristics we found that make applications more suitable for PNM are large memory footprint, small FLOPS and very low arithmetic intensity. For PUM, a large memory footprint leads to higher speedup but also more energy consumption.

The advantage of our simulated PNM architecture is very small memory read latency while the disadvantage is the low frequency of the PNM core. Since the core is slower than CPU and nearer to memory, PNM may also have an advantage in energy consumption. PUM is a powerful emerging technology that can provide great improvement of latency compared to CPU. The drawbacks such as scalability with input size also limit the actual implementation of PUM.

We have found a way to apply PNM processing on CNN acceleration by application partition. We partitioned the CNN models into layers. According to the characteristics of the layer, we assigned it to PNM architecture or CPU. Due to the time limitation, we have not found a way to combine PNM and PUM togther.

By using current simulation flow and tools, there is still much work that can be done in the future. First, now we have only four CNN models. The application characterization may be not comprehensive. Second, except for latency, energy is another aspect of performance. We have done the energy analysis of VGG8 using McPAT. The same analysis can be done on other models. Finally, we have compared PNM with CPU and compared PUM with CPU. The next step is to compare PNM with PUM. This will provide us with a deeper understanding of PIM technologies.

Bibliography

[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct 2017. [Online]. Available: https://doi.org/10.1038/nature24270

[2] OpenAI, M. Andrychowicz, B. Baker, M. Chociej, R. Józefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning Dexterous In-Hand Manipulation," *CoRR*, vol. abs/1808.00177, 2018. [Online]. Available: http://arxiv.org/abs/1808.00177

[3] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for YouTube Recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*, ser. RecSys '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 191–198. [Online]. Available: https://doi.org/10.1145/2959100.2959190

[4] C.-Y. Wu, A. Ahmed, A. Beutel, A. J. Smola, and H. Jing, "Recurrent Recommender Networks," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 495–503. [Online]. Available: https://doi.org/10.1145/3018661.3018689

[5] M. Himmelsbach, T. Luettel, and H.-J. Wuensche, "Real-time object classification in 3d point clouds using point feature histograms," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 994–1000.

[6] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in Perception for Autonomous Driving: Waymo Open Dataset," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 2443–2451.

[7] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological review*, vol. 65 6, pp. 386–408, 1958. [Online]. Available: https://api.semanticscholar.org/CorpusID:12781225

[8] 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, 2022.

[9] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and Benchmarking of Machine Learning Accelerators," in *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, 2019, pp. 1–9.

[10] J. Nickolls and W. J. Dally, "The GPU Computing Era," *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010.

[11] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU Computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[12] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[13] K. E. A. van de Sande, T. Gevers, and C. G. M. Snoek, "Empowering Visual Categorization with the GPU," *IEEE Transactions on Multimedia*, vol. 13, no. 1, pp. 60–70, 2011.

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[15] Nvidia, "CUDA Toolkit Documentation 12.5 Update 1," https://docs.nvidia.com/cuda/.

[16] ——, "Developer Guide - Overview," https://docs.nvidia.com/deeplearning/cudnn/latest/developer/overview.html.

[17] Open GPU Data Science, "RAPIDS," https://github.com/orgs/rapidsai.

[18] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933118302291

[19] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, "GPUs and the Future of Parallel Computing," *IEEE Micro*, vol. 31, no. 5, pp. 7–17, 2011.

[20] X. Mei and X. Chu, "Dissecting GPU Memory Hierarchy through Microbenchmarking," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 72–86, 2017.

[21] G. Kestor, R. Gioiosa, D. J. Kerbyson, and A. Hoisie, "Quantifying the energy cost of data movement in scientific applications," in *2013 IEEE International Symposium on Workload Characterization (IISWC)*, 2013, pp. 56–65.

[22] D. Pandiyan and C.-J. Wu, "Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014, pp. 171–180.

[23] A. Gebregiorgis, H. A. Du Nguyen, J. Yu, R. Bishnoi, M. Taouil, F. Catthoor, and S. Hamdioui, "A Survey on Memory-centric Computer Architectures," *J. Emerg. Technol. Comput. Syst.*, vol. 18, no. 4, oct 2022. [Online]. Available: https://doi.org/10.1145/3544974

[24] J. T. Pawlowski, "Hybrid memory cube (HMC)," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, 2011, pp. 1–24.

[25] H. Jun, J. Cho, K. Lee, H.-Y. Son, K. Kim, H. Jin, and K. Kim, "HBM (High Bandwidth Memory) DRAM Technology and Architecture," in *2017 IEEE International Memory Workshop (IMW)*, 2017, pp. 1–4.

[26] J. Gómez-Luna, I. El Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu, "Benchmarking Memory-Centric Computing Systems: Analysis of Real Processing-In-Memory Hardware," in *2021 12th International Green and Sustainable Computing Conference (IGSC)*, 2021, pp. 1–7.

[27] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, J. Lee, D. Ko, Y. Jun, K. Cho, I. Kim, C. Song, C. Jeong, D. Kwon, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[28] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H.-S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E.-B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "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 *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 350–352.

[29] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 27–39.

[30] N. Lepri, P. Gibertini, P. Mannocci, A. Pirovano, I. Tortorelli, P. Fantini, and D. Ielmini, "In-memory neural network accelerator based on phase change memory (PCM) with one-selector/one-resistor (1S1R) structure operated in the subthreshold regime," in *2023 IEEE International Memory Workshop (IMW)*, 2023, pp. 1–4.

[31] C. Li, Y. Yang, M. Feng, S. Chakradhar, and H. Zhou, "Optimizing Memory Efficiency for Deep Convolutional Neural Networks on GPUs," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 633–644.

[32] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[33] J. Schmidhuber, "Deep Learning in Neural networks: An Overview," *Neural Networks*, vol. 61, pp. 85–117, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135

[34] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[35] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990. [Online]. Available: https://www.sciencedirect.com/science/article/pii/036402139090002E

[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010.

[37] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[38] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu, Z. Yang, Y. Zhang, and D. Tao, "A Survey on Vision Transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 1, pp. 87–110, 2023.

[39] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a Convolutional Neural Network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[40] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[41] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, pp. 3212–3232, 2019.

[42] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[43] X. Fu, J. Huang, X. Ding, Y. Liao, and J. Paisley, "Clearing the Skies: A Deep Network Architecture for Single-Image Rain Removal," *IEEE Transactions on Image Processing*, vol. 26, no. 6, pp. 2944–2956, 2017.

[44] S. Ji, W. Xu, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[45] D. Maturana and S. Scherer, "VoxNet: A 3D Convolutional Neural Network for real-time object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[46] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding Deep Neural Networks with Rectified Linear Units," *CoRR*, vol. abs/1611.01491, 2016. [Online]. Available: http://arxiv.org/abs/1611.01491

[47] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[48] J. Hu, L. Shen, and G. Sun, "Squeeze-and-Excitation Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132–7141.

[49] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1314–1324.

[50] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:195908774

[52] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, 2015, pp. 730–734.

[53] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[54] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[55] V. Sze, Y. Chen, J. S. Emer, A. Suleiman, and Z. Zhang, "Hardware for Machine Learning: Challenges and Opportunities," *CoRR*, vol. abs/1612.07625, 2016. [Online]. Available: http://arxiv.org/abs/1612.07625

[56] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 92–104.

[57] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," in *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 696–701.

[58] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[59] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, "A Survey of FPGA Based Neural Network Accelerator," *CoRR*, vol. abs/1712.08934, 2017. [Online]. Available: http://arxiv.org/abs/1712.08934

[60] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, "Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 3, jul 2017. [Online]. Available: https://doi.org/10.1145/3079758

[61] X. Zhang, X. Liu, A. Ramachandran, C. Zhuge, S. Tang, P. Ouyang, Z. Cheng, K. Rupnow, and D. Chen, "High-performance video content recognition with long-term recurrent convolutional network for FPGA," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–4.

[62] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," *CoRR*, vol. abs/1704.04760, 2017. [Online]. Available: http://arxiv.org/abs/1704.04760

[63] A. Boroumand, S. Ghose, B. Akin, R. Narayanaswami, G. F. Oliveira, X. Ma, E. Shiu, and O. Mutlu, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021, pp. 159–172.

[64] H. S. Stone, "A Logic-in-Memory Computer," *IEEE Transactions on Computers*, vol. C-19, no. 1, pp. 73–78, 1970.

[65] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 105–117.

[66] C. Xie, S. L. Song, J. Wang, W. Zhang, and X. Fu, "Processing-in-Memory Enabled Graphics Processors for 3D Rendering," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 637–648.

[67] M. Zhou, M. Imani, S. Gupta, Y. Kim, and T. Rosing, "GRAM: Graph Processing in a ReRAM-Based Computational Memory," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 591–596. [Online]. Available: https://doi.org/10.1145/3287624.3287711

[68] B. Akin, F. Franchetti, and J. C. Hoe, "Data reorganization in memory using 3D-stacked DRAM," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 131–143.

[69] K. Hsieh, S. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3D-stacked memory: Challenges, mechanisms, evaluation," in *2016 IEEE 34th International Conference on Computer Design (ICCD)*, 2016, pp. 25–32.

[70] A. Augusta and S. Idreos, "JAFAR: Near-Data Processing for Databases," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 2069–2070. [Online]. Available: https://doi.org/10.1145/2723372.2764942

[71] J. H. Lee, J. Sim, and H. Kim, "BSSync: Processing Near Memory for Machine Learning Workloads with Bounded Staleness Consistency Models," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 241–252.

[72] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14–26.

[73] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," *SIGPLAN Not.*, vol. 53, no. 2, p. 316–331, mar 2018. [Online]. Available: https://doi.org/10.1145/3296957.3173177

[74] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "A Modern Primer on Processing in Memory," in *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Singapore: Springer Nature Singapore, 2023, pp. 171–243. [Online]. Available: https://doi.org/10.1007/978-981-16-7487-7_7

[75] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, "25.2 A 1.2V 8Gb 8-channel 128GB/s

high-bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using 29nm process and TSV," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 432–433.

[76] D. U. Lee, K. W. Kim, K. W. Kim, K. S. Lee, S. J. Byeon, J. H. Kim, J. H. Cho, J. Lee, and J. H. Chun, "A 1.2 V 8 Gb 8-Channel 128 GB/s High-Bandwidth Memory (HBM) Stacked DRAM With Effective I/O Test Circuits," *IEEE Journal of Solid-State Circuits*, vol. 50, no. 1, pp. 191–203, 2015.

[77] C. Giannoula, I. Fernandez, J. Gómez-Luna, N. Koziris, G. Goumas, and O. Mutlu, "SparseP: Towards Efficient Sparse Matrix Vector Multiplication on Real Processing-In-Memory Systems," *CoRR*, vol. abs/2201.05072, 2022. [Online]. Available: https://arxiv.org/abs/2201.05072

[78] G. F. Oliveira, J. Gómez-Luna, S. Ghose, A. Boroumand, and O. Mutlu, "Accelerating Neural Network Inference With Processing-in-DRAM: From the Edge to the Cloud," *IEEE Micro*, vol. 42, no. 6, pp. 25–38, 2022.

[79] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, J. Chen, J. Chen, Z. Chen, M. Chrzanowski, A. Coates, G. Diamos, K. Ding, N. Du, E. Elsen, J. Engel, W. Fang, L. Fan, C. Fougner, L. Gao, C. Gong, A. Hannun, T. Han, L. V. Johannes, B. Jiang, C. Ju, B. Jun, P. LeGresley, L. Lin, J. Liu, Y. Liu, W. Li, X. Li, D. Ma, S. Narang, A. Ng, S. Ozair, Y. Peng, R. Prenger, S. Qian, Z. Quan, J. Raiman, V. Rao, S. Satheesh, D. Seetapun, S. Sengupta, K. Srinet, A. Sriram, H. Tang, L. Tang, C. Wang, J. Wang, K. Wang, Y. Wang, Z. Wang, Z. Wang, S. Wu, L. Wei, B. Xiao, W. Xie, Y. Xie, D. Yogatama, B. Yuan, J. Zhan, and Z. Zhu, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16. JMLR.org, 2016, p. 173–182.

[80] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[81] G. Singh, L. Chelini, S. Corda, A. Javed Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, "A Review of Near-Memory Computing Architectures: Opportunities and Challenges," in *2018 21st Euromicro Conference on Digital System Design (DSD)*, 2018, pp. 608–617.

[82] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "RowClone: Fast and energy-efficient in-DRAM bulk data copy and initialization," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 185–197.

[83] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 273–287.

[84] M. Poremba and Y. Xie, "NVMain: An Architectural-Level Main Memory Simulator for Emerging Non-volatile Memories," in *2012 IEEE Computer Society Annual Symposium on VLSI*, 2012, pp. 392–397.

[85] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase Change Memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, 2010.

[86] Y. Chen, "ReRAM: History, Status, and Future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[87] L. Song, X. Qian, H. Li, and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 541–552.

[88] G. W. Burr, P. Narayanan, S. Ambrogio, A. Okazaki, H. Tsai, K. Hosokawa, C. Mackin, A. Nomura, T. Yasuda, J. Demarest, K. W. Brew, V. Chan, S. Choi, T. Gordon, T. M. Levin, A. Friz, M. Ishii, Y. Kohda, A. Chen, A. Fasoli, J. Luquin, N. Saulnier, S. Teehan, I. Ahsan, and V. Narayanan, "Phase Change Memory-based Hardware Accelerators for Deep Neural Networks (invited)," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2023, pp. 1–2.

[89] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L.-Y. Chen, B. Zhang, and P. Deaville, "In-Memory Computing: Advances and Prospects," *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.

[90] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian, and X. Li, "PIMSim: A Flexible and Detailed Processing-in-Memory Simulator," *IEEE Computer Architecture Letters*, vol. 18, no. 1, pp. 6–9, 2019.

[91] 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, 2016.

[92] Samsung Advanced Institute of Technology, "PIMsimulator," https://github.com/SAITPublic/PIMSimulator, 2021.

[93] C. Yu, S. Liu, and S. Khan, "MultiPIM: A Detailed and Configurable Multi-Stack Processing-In-Memory Simulator," *IEEE Computer Architecture Letters*, vol. 20, no. 1, pp. 54–57, 2021.

[94] X. Peng, S. Huang, Y. Luo, X. Sun, and S. Yu, "DNN+NeuroSim: An End-to-End Benchmarking Framework for Compute-in-Memory Accelerators with Versatile Device Technologies," in *2019 IEEE International Electron Devices Meeting (IEDM)*, 2019, pp. 32.5.1–32.5.4.

[95] C. Karthik, W. Christian, Y. Li, S. Goossens, M. Jung, O. Naji, B. Akesson, N. Wehn, and K. Goossens, "DRAMPower: Open-source DRAM Power & Energy Estimation Tool," https://http://www.drampower.info.

[96] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO 42: Proceedings of the*

42nd Annual IEEE/ACM International Symposium on Microarchitecture, 2009, pp. 469–480.

[97] G. Singh, J. Gómez-Luna, G. Mariani, G. F. Oliveira, S. Corda, S. Stuijk, O. Mutlu, and H. Corporaal, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[98] M. Harris, "Mapping computational concepts to GPUs," in *ACM SIGGRAPH 2005 Courses*, ser. SIGGRAPH '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 50–es. [Online]. Available: https://doi.org/10.1145/1198555.1198768

[99] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

A

Appendix

Table A.1: Metrics of VGG8

layers	input size	kernel size	parameters	arithmetic intensity
conv1	[1,3,32,32]	[3,128,3,3]	6528	1084.24
conv2	[1,128,32,32]	[128,128,3,3]	278528	1084.24
conv3	[1,128,16,16]	[128,256,3,3]	327680	460.80
conv4	[1,256,16,16]	[256,256,3,3]	655360	460.80
conv5	[1,256,8,8]	[256,512,3,3]	1196032	126.25
conv6	[1,512,8,8]	[512,512,3,3]	2392064	126.25
fc1	[1,8192]	[8192,1024]	8388608	2.00
fc2	[1,1024]	[1024,10]	10240	1.82

Table A.2: Metrics of UNet

layers	input size	kernel size	parameters	arithmetic intensity
conv2	[1,3,32,32]	[3,3,3,3]	3153	52.61
conv4	[1,6,16,16]	[6,6,3,3]	1860	89.19
conv6	[1,12,8,8]	[12,12,3,3]	2064	80.37
conv8	[1,24,4,4]	[24,24,3,3]	5568	29.79
conv12	[1,24,4,4]	[24,24,3,3]	5568	29.79
conv14	[1,12,8,8]	[12,12,3,3]	2064	80.37
conv16	[1,6,16,16]	[6,6,3,3]	1860	89.19
conv18	[1,3,32,32]	[3,3,3,3]	3153	52.61
conv19	[1,3,32,32]	[3,2,1,1]	3078	3.99

Table A.3: Metrics of ResNet

layers	input size	kernel size	parameters	arithmetic intensity
conv1	[1,64,56,56]	[64,64,3,3]	237568	973.24
conv2	[1,64,56,56]	[64,64,3,3]	237568	973.24
downsample1	[1,128,28,28]	[64,128,1,1]	108544	118.34
downsample2	[1,256,14,14]	[128,256,1,1]	82944	154.94
fc	[1,512]	[512,1000]	512512	2.00

Table A.4: Metrics of MobileNet

layers	input size	kernel size	parameters	arithmetic intensity
conv1	[1,96,7,7]	[96,576,1,1]	60000	90.34
conv2	[1,576,7,7]	[1,576,5,5]	426624	33.14
conv3	[1,576,7,7]	[576,96,1,1]	835520	64.89
SE.conv1	[1, 96,1,1]	[96,24,1,1]	2400	1.94
SE.conv2	[1, 24,1,1]	[24,96,1,1]	2328	2.06
fc1	[1,28224]	[28224,1280]	36154944	2.00
fc2	[1,1280]	[1280,1000]	129280	1.98