Optimizing Convolutional Neural Networks on Processing-in-Memory Architectures: Implementation, Benchmarking, and Performance Analysis

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Hugo Abbot

Spring, 2025

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Kevin Skadron, Department of Computer Science

Technical Report

I. Introduction

The increasing demand for high-performance and energy-efficient data processing has exposed critical limitations in traditional computing architectures, particularly in data-intensive and increasingly used applications like machine learning and artificial intelligence. Central processing units (CPUs) and graphics processing units (GPUs) have historically served as the backbone of computation; however, their reliance on discrete memory access causes significant data movement overhead (Jacob, 2010). This occurrence – known as the von Neumann bottleneck – limits system performance and contributes substantially to energy consumption. As the returns on transistor scaling diminish and Moore's Law slows down, novel architectural paradigms have become necessary (Moore, 1998).

One such paradigm is Processing in Memory (PIM), which aims to address the data movement bottleneck by integrating computational capabilities directly into or near memory. By eliminating the need to shuttle data back and forth between the processor and memory, PIM architectures provide significant gains in both performance and energy efficiency for certain computational capacities (Stone, 1970). Recent research has demonstrated that PIM can achieve notable speedups in workloads that are memory-bound, such as scientific computing, graph processing, and increasingly, machine learning (Gómez-Luna, 2023; Blackford, 2002).

Convolutional Neural Networks (CNNs) are a particularly promising application domain for PIM, due to their widespread use in image recognition, natural language processing, and autonomous systems. In addition, CNNs' computational structure, which often includes highly parallelizable operations such as convolutions and pooling, also provides some promise with regards to performance when used with PIM. In this project, we explore the feasibility and

effectiveness of mapping key CNN kernels – specifically those from the ResNet-18 architecture – onto PIM systems using a high-level C++ simulator and benchmark suite developed by The Laboratory for Computer Architecture at Virginia (LAVA Lab).

II. The Problem

Despite the theoretical advantages of PIM, practical implementation and performance optimization for specific workloads remain a challenge. For instance, CNNs are composed of a sequence of diverse computational kernels – including convolutions, activations like ReLU, and pooling like max and average – each with their own unique memory and computational requirements. The goal of this project is to evaluate the performance of key CNN kernels when executed on general-purpose PIM architectures, with a focus on identifying which components of ResNet-18 can most efficiently be mapped to PIM. This includes the testing of various kernels using our custom PIM simulation API and benchmarking them against CPU and GPU baselines using optimized libraries such as PyTorch.

In the latter stages of this work, additional focus has been placed on optimizing convolution layers, which are among the most computationally intensive components of CNNs. Using batching strategies and parallelism within our simulator, we aim to assess the performance scaling of convolution operations on PIM systems, exploring the behavior of end-to-end implementations like ResNet and VGG for better analysis.

Ultimately, this work contributes to the broader objective of hybrid execution models in machine learning. By determining which kernels perform best on which type of processor, we envision a heterogeneous computing approach where models are dynamically partitioned across compute units for optimal performance. This approach not only advances our understanding of

PIM capabilities but also pushes toward more intelligent workload scheduling and resource utilization in next-generation AI hardware systems.

III. The Novelty

While prior work has demonstrated speedups for general machine learning operations on PIM architectures, there is a lack of detailed analysis regarding the granular performance of individual CNN kernels across different hardware platforms, including PIM (Roy et al., 2021). This project distinguishes itself by looking into the kernel-level analysis of these components, finding which operations might benefit most from PIM architectures and which do not.

By integrating these insights into an end-to-end implementation of ResNet-18, this work bridges the gap between low-level kernel performance and high-level model behavior. This layered approach not only enables a deeper understanding of PIM's strengths and limitations but also informs hybrid execution strategies based on empirical performance. To our knowledge, this level of analysis within full model benchmarks on simulated PIM architecture has not been previously done, making this a valuable contribution to the field of PIM for deep learning.

IV. Our Approach

The core objective of this project was to design and implement an end-to-end benchmark of the ResNet-18 model with our custom PIM simulator. While the LAVA Lab's existing support for CNN architectures – most notably the VGG13, VGG16, and VGG19 models used in our IISWC 2024 submission – provided a structural and code-level foundation, ResNet-18 introduced architectural challenges that necessitated significant extensions to the framework (Siddique et al., 2024).

Unlike the strictly sequential nature of the VGG family, ResNet-18 is characterized by the use of residual learning through skip connections, a concept introduced in previous work (He

et al., 2015). These residual blocks, also called building blocks, allow the network to learn residual functions relative to the input, which greatly improves gradient flow and model convergence in deeper networks. Each building block typically consists of one convolution block and one identity block, each of which with two convolutional layers interleaved with ReLU activations. The two subset blocks differ in whether the shortcut path that creates the residual learning does so by directly adding the input after a ReLU activation to the output – as in the case of the identity block – or applying a 1x1 convolution to the input and then adding it to the output – as in the case of the convolution block. This architecture is depicted in Figure 1 and depicts the second building block within the ResNet-18 CNN, with Figure 2 representing the entire structure of the ResNet-18 model.

Figure 1

Example building block within a ResNet neural network (Modi, 2021)

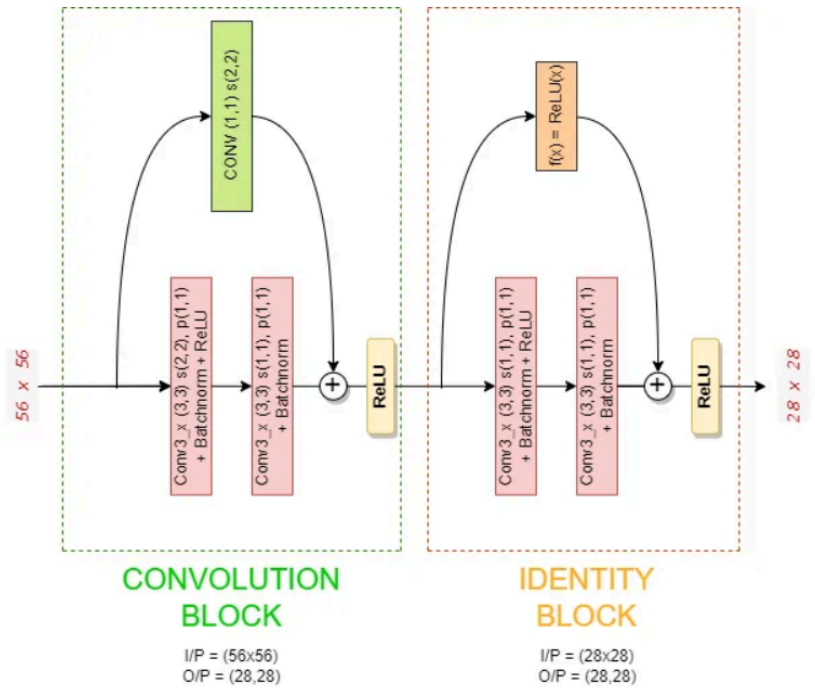


Figure 2

Overall ResNet-18 architecture by layer

	Layer	Input	Weight	Output	Layer	Input	Weight	Output	
	Conv 1-1	224x224x3	7x7x64 (s2)	112x112x64	Conv 4-1	28x28x128	3x3x256 (s2)	14x14x256	BB3
	Max pool	112x112x64	NA (s2)	56x56x64	Conv 4-2	14x14x256	3x3x256	14x14x256	
BB1	Conv 2-1	56x56x64	3x3x64	56x56x64	Conv 4-3	14x14x256	3x3x256	14x14x256	
	Conv 2-2	56x56x64	3x3x64	56x56x64	Conv 4-4	14x14x256	3x3x256	14x14x256	
	Conv 2-3	56x56x64	3x3x64	56x56x64	Conv 5-1	14x14x256	3x3x512 (s2)	7x7x512	
	Conv 2-4	56x56x64	3x3x64	56x56x64	Conv 5-2	7x7x512	3x3x512	7x7x512	BB4
	Conv 3-1	56x56x64	3x3x128 (s2)	28x28x128	Conv 5-3	7x7x512	3x3x512	7x7x512	
BB2	Conv 3-2	28x28x128	3x3x128	28x28x128	Conv 5-4	7x7x512	3x3x512	7x7x512	
	Conv 3-3	28x28x128	3x3x128	28x28x128	Global Avg Pool	7x7x512	NA	1x1x512	
	Conv 3-4	28x28x128	3x3x128	28x28x256	Dense	1x512	512x1000	1X1000	

Note. There is a ReLU activation after each individual convolution layer

To accurately replicate this architecture within our custom high-level simulator, every layer of the ResNet-18 model was manually defined using the suite's C++ API. This involved implementing convolutional layers, ReLU activations, batch normalization, max pooling, and global average pooling – each called individually and explicitly, allowing for easier understanding and better detailed analysis. Most attention during development was given to the construction of the residual blocks, where conditional logic was introduced to handle identity and projection shortcuts as required by the model's topology. Due to this being the first introduction of the residual mechanism into our benchmarking, much verification was performed to ensure correct and reliable execution.

The simulator's API abstracts away low-level memory and compute behavior, offering a high-level interface for performance benchmarking on PIM architectures. Over 1,100 lines of C++ code were written in order to carefully construct and orchestrate each layer and block modularly, enabling precise control over execution. This manual, component-by-component construction not only ensured faithfulness to the original proposed research paper model but also

allowed us to isolate and measure the performance and validation of individual kernels under various simulations. Various performance metrics can be tracked with our simulator, such as architecture-specific configurations (i.e. PIM core layout, DRAM structure), data movement metrics, per-operation stats, and total estimated runtime and energy usage. All of this information is crucial in determining bottlenecks with the current implementation and how future updates, such as a non-host based convolution, might provide performance improvements.

By extending the benchmarking suite to support ResNet-18, this work not only broadened the simulator's coverage of modern CNN architectures but also provided a reusable template for implementing and evaluating other residual-style or transformer-based models in future research. The benchmark now serves as both a performance validation tool and an experimental platform for exploring workload partitioning across heterogeneous systems.

V. Our Findings

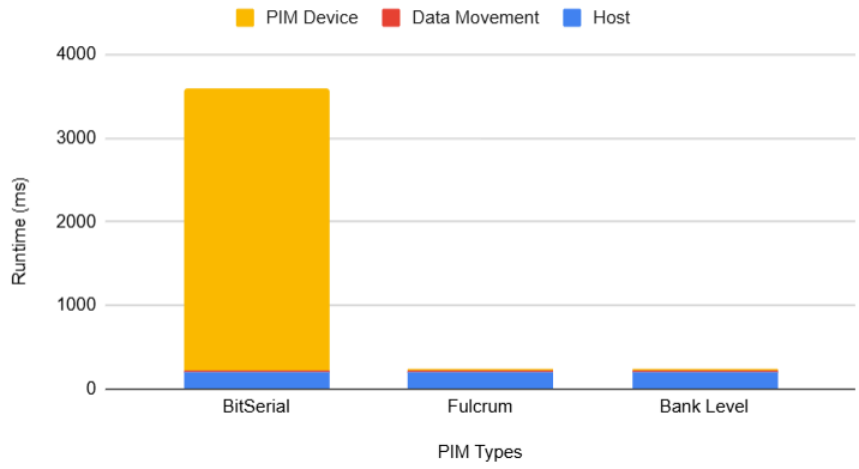
The performance evaluation of ResNet-18 across various PIM architectures reveals notable disparities in runtime and operational efficiency. Among the simulated configurations, bit-parallel subarray-level PIM demonstrated the best overall performance, followed closely by bank-level PIM, while bit-serial PIM lagged significantly behind. Bit-serial PIM incurred nearly a 175x slowdown in total execution time when compared to bit-parallel, largely due to its bit-by-bit processing approach and its quadratic complexity with respect to operand bit width. This performance bottleneck became especially evident during convolutional operations, which are inherently multiplication-heavy and particularly taxing under a bit-serial computation model.

To provide quantitative context, the bit-parallel PIM configuration completed the ResNet-18 benchmark in 236.38 ms, the bank-level PIM version required 246.17 ms, while the bit-serial took a much longer time at 3,585.59 ms. However, these runtimes, which are

represented in Figure 3, include more than just in-memory computation – they also account for data movement overhead and host-side execution components, which represent a substantial fraction of total time for the two fastest configurations. Specifically, only 18.48 ms of the bit-parallel runtime and 28.27 ms of the bank-level runtime were attributed to PIM-based computation. The remaining runtime was spent on data transfer between the host and the memory device, as well as host-based processing tasks that are currently not handled entirely within the PIM subsystem. These include components of the convolution layers which, in the present implementation, are split between PIM and the host processor. Unlike the other two, bit-serial was different as nearly 94% of the total execution time was spent on the PIM computation, showing how nonoptimal its use is within this current context.

Figure 3

ResNet-18 performance chart with a PIM configuration of 16 ranks, 128 banks per rank, 16 subarrays per bank, 2,048 rows per subarray, and 8,192 columns per subarray.



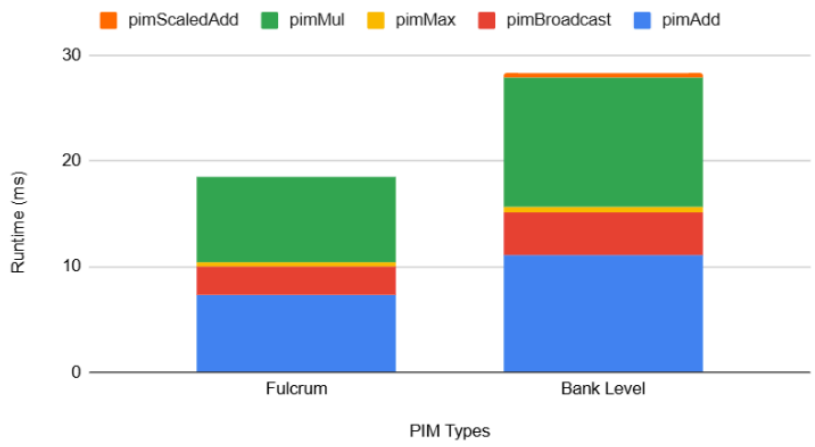
Note. Fulcrum is a proposed version of bit-parallel subarray-level PIM (Lenjani et al., 2020).

A breakdown of operation-level statistics highlights the computational hotspots within the model. Shown in Figure 4, both bit-parallel and bank-level PIM most frequently invoke

32-bit integer multiplication and addition operations, which were called 38,656 and 35,712 times respectively. These operations are predominantly tied to the convolution layers, which dominate the computational cost of CNNs. The high frequency of these calls reinforces the need for optimizing convolutional routines in order to improve overall performance.

Figure 4

PIM operation breakdown between bit-parallel and bank-level PIM architectures



In comparison to traditional compute platforms, the current PIM implementations still fall short. The CPU baseline completed the same benchmark in 231.46 ms, which is slightly faster than either PIM configuration when total execution time is considered. While bit-parallel PIM outperforms CPU in raw in-memory compute time, the benefits are effectively neutralized when the overhead from data transfers and host-side operations is included. The GPU baseline, by contrast, achieved a significantly lower execution time of 2.50 ms, underscoring the advantage of highly parallelized hardware and optimized compute pipelines in existing accelerators.

VI. Future Work

To overcome current performance limitations, the lab has begun developing an optimized, fully in-PIM convolution implementation. Unlike the initial version, which offloads parts of the convolutional computation to the host, this new design executes 100% of the kernel

operations within the PIM object itself using only in-memory operations. Preliminary results from this new implementation show promising gains, achieving up to 10x speedup over the prior PIM convolution version. This change significantly reduces data movement and eliminates host-side overhead, making PIM performance far more competitive – especially when compared to CPU, though it still falls short of GPU performance.

We expect this optimization to significantly shift the balance in hybrid execution strategies. With fully in-PIM kernels like convolution, it becomes feasible to delegate larger portions of the model to PIM and rely less on CPU and GPU interaction. In the future, we also plan to re-evaluate performance using more sophisticated data batching techniques. These improvements will not only improve standalone PIM efficiency but also inform intelligent workload partitioning strategies for heterogeneous ML acceleration platforms.

VII. Conclusion

This project investigated the performance of CNN workloads, specifically ResNet-18, on a range of simulated Processing-in-Memory architectures. Through detailed kernel-level benchmarking and analysis, we demonstrated that while PIM offers clear computational advantages in isolation, real-world performance remains limited by host-side execution and data movement overhead. Bit-parallel subarray-level and bank-level PIM showed the most promise, but even these configurations could not outperform CPU baselines when considering total runtime, and were far outpaced by GPU acceleration.

However, our development of a fully in-PIM convolution kernel suggests a path forward. Early results indicate substantial speedups by eliminating host interaction, supporting the case for deeper integration of in-memory compute in ML workloads. These findings lay the groundwork for future hybrid execution strategies, where intelligently partitioned workloads

could leverage the strengths of PIM, CPU, and GPU in tandem to achieve optimal performance and energy efficiency.

References

Blackford, I. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., & et al. (2002). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software (TOMS)*, *Volume 28*, pp. 135-151.

Gómez-Luna, J., Guo, Y., Brocard, S., Legriel, J., Cimadomo, R., & Oliveira, G. F. (2023). Evaluating Machine Learning Workloads on Memory-Centric Computing Systems. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 35-49. https://ieeexplore.ieee.org/document/10158216

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. https://arxiv.org/pdf/1512.03385

Jacob, B., Wang, D., & Ng, S. (2010). *Memory systems: cache, DRAM, disk*.

Lenjani, M., Gonzalez, P., Sadredini, E., Li, S., Xie, Y., Akel, A., Eilert, S., Stan, M. R., & Skadron, K. (2020). Fulcrum: A Simplified Control and Access Mechanism Toward Flexible and Practical In-Situ Accelerators. *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. https://doi.org/10.1109/hpca47549.2020.00052

Modi, R. (2021, December 1). *ResNet — Understand and Implement from scratch*. Analytics Vidhya; Medium. https://medium.com/analytics-vidhya/resnet-understand-and-implement-from-scratch-d0eb9725e0db

Moore, G. E. (1998). Cramming more components onto integrated circuits. *Proceedings of the IEEE*, *Volume 86*, pp. 82-85.

Roy, S., Ali, M., & Raghunathan, A. (2021). *PIM-DRAM: Accelerating Machine Learning Workloads using Processing in Commodity DRAM*. ArXiv.org. https://arxiv.org/abs/2105.03736

Siddique, F. A., Guo, D., Fan, Z., Gholamrezaei, M., Baradaran, M., Ahmed, A., Abbot, H., Durrer, K., Nandagopal, K., Ermovick, E., Kiyawat, K., Gul, B., Mughrabi, A., Venkat, A., & Skadron, K. (2024). Architectural Modeling and Benchmarking for Digital DRAM PIM. *IEEE International Symposium on Workload Characterization (IISWC)*. http://www.cs.virginia.edu/~skadron/Papers/PIMbench_PIMeval_iiswc2024.pdf

Stone, H. S. (1970). A logic-in-memory computer,. *IEEE Transactions on Computers (TC)*, *Volume 100*, pp. 73-78.