PKU-IAI Technical Report: PKU-IAI-2022-T-0001

Behavioral Modeling and Performance Analysis of CNN Inference on CIM Architecture

Sicheng Hu
Institute of Artificial Intelligence
Peking University
husicheng@stu.pku.edu.cn

Abstract

The physical separation of processing and storage units in conventional Von Neumann architectures imposes severe latency and energy penalties on data-intensive neural network workloads. Computing-in-Memory (CIM) technologies mitigate these inefficiencies by executing Matrix-Vector Multiplication (MVM) in situ, though typically at the cost of introducing analog computational noise. This report examines the deployment of the VGG16 architecture within a **behavioral-level CIM simulation framework** to characterize the trade-off between hardware-induced error and algorithmic performance. Simulation results indicate that the model exhibits high resilience against analog noise in this setup. Despite a substantial rise in Mean Squared Error (MSE) within intermediate feature maps, the directional alignment of feature vectors remains largely intact, ensuring that the final classification accuracy remains stable under tested noise intensities. This observed disparity suggests that the decision boundaries of such over-parameterized classifiers are sufficiently robust to tolerate the stochastic deviations inherent in analog computation. These findings imply that the strict replication of floating-point arithmetic might be relaxed for successful inference in specific resource-constrained edge environments. The source code and demonstration video of this implementation are provided at `https://github.com/Hsch22/VGG16-CIM-Sim`.

1 Introduction

Convolutional Neural Networks (CNNs) have established themselves as the dominant framework for computer vision, driving breakthroughs in tasks such as image classification and object detection [7]. Despite this success, the deployment of such computationally intensive models on edge devices presents a significant engineering challenge. The fundamental limitation lies in the traditional Von Neumann architecture, which physically separates the processing unit from the memory unit [1]. This separation necessitates frequent and extensive data shuttling across the system bus.

This architectural constraint manifests as the “Memory Wall,” a bottleneck where the latency and energy consumption of data movement vastly exceed the cost of arithmetic computation itself [10]. Furthermore, studies have quantified that memory access energy can be orders of magnitude higher than that of floating-point operations [4]. To mitigate these inefficiencies, CIM has emerged as a promising paradigm. By performing analog or digital computation directly within the memory arrays, CIM minimizes data transport, thereby offering substantial improvements in both energy efficiency and system throughput.

For workloads dominated by matrix-vector multiplications, CIM designs demonstrate orders-of-magnitude improvements in throughput and efficiency over conventional GPUs. As shown in Figure 1, AIG-CIM [5] outperforms the RTX 3090 by over $20\times$ in latency and $200\times$ in throughput. These gains stem from a tri-gear heterogeneous architecture tailored to U-Net's weight reuse and computational profiles. By optimally mapping diverse workloads to specialized macros, this design maximizes utilization, validating CIM as a scalable solution for next-generation AI.

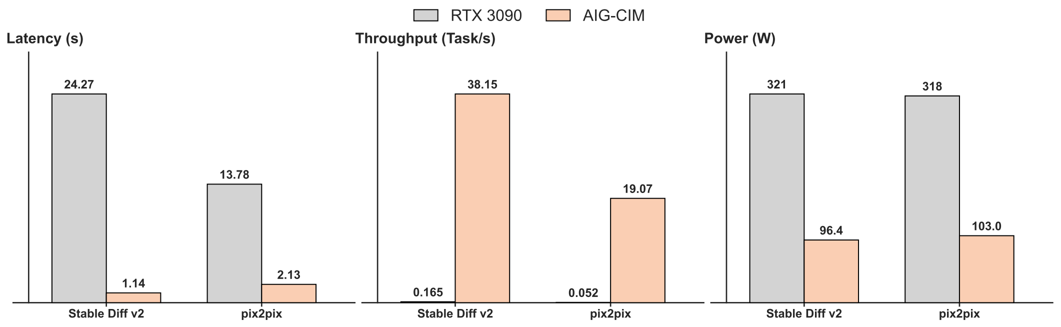

Figure 1: Performance comparison between AIG-CIM and RTX 3090 GPU across two diffusion models.

To gain a deeper insight into the hardware-software co-design implications, a representative CNN architecture, VGG16, is deployed onto a simulated CIM platform. This study analyzes MVM mapping strategies and quantifies the impact of quantization and thermal noise on inference accuracy.

2 Literature Review

2.1 Motivation: The Von Neumann Bottleneck

In conventional digital accelerators (e.g., GPUs and ASICs), the energy cost of fetching data from off-chip DRAM differs by orders of magnitude from that of floating-point arithmetic [4]. This disparity creates severe inefficiencies for data-intensive deep learning workloads. CIM architectures address this by co-locating storage and computation, thereby reducing the latency and energy overheads of weight data movement [8].

2.2 Principle of Computing-in-Memory

CIM architectures rely on crossbar arrays, typically implemented using non-volatile memory (e.g., ReRAM [3], PCM [6]) or modified SRAM. These arrays leverage analog circuit laws to execute MVM in parallel. In this paradigm, network weights (W) are mapped to cell conductances (G), while input activations (X) are converted to analog voltages (V) applied to wordlines. According to Ohm's Law and Kirchhoff's Current Law (KCL), the current accumulated along each bitline represents the dot product:

$$I_{out,j} = \sum_i V_{in,i} \cdot G_{i,j} \tag{1}$$

As illustrated in Figure 2a, these analog currents I_{out} are subsequently digitized via Analog-to-Digital Converters (ADCs) for further processing [9].

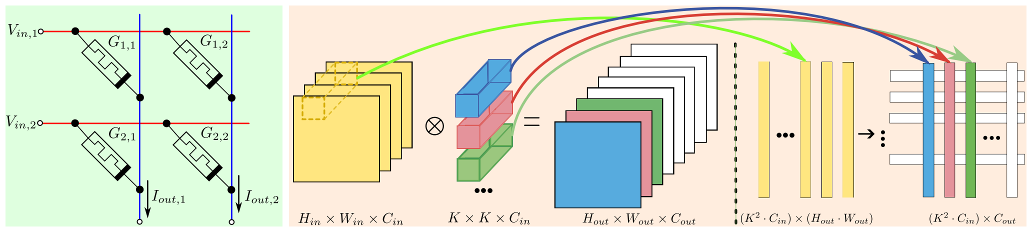
$V_{in,1}$
$G_{1,1}$
$G_{1,2}$
$V_{in,2}$
$G_{2,1}$
$G_{2,2}$
$I_{out,1}$
$I_{out,2}$
$H_{in} \times W_{in} \times C_{in}$
$\otimes$
$K \times K \times C_{in}$
$=$
$H_{out} \times W_{out} \times C_{out}$
$(K^2 \cdot C_{in}) \times (H_{out} \cdot W_{out})$
$(K^2 \cdot C_{in}) \times C_{out}$
(a)
(b)

Figure 2: Hardware implementation principles for VGG16. (a) Schematic of a resistive crossbar array performing analog MVM. (b) Workflow of deploying a Convolutional Layer onto CIM, illustrating the *im2col* transformation and weight mapping strategy.

2.3 Mapping CNNs to CIM Architectures

To bridge the architectural gap between convolutional data flow and the physical constraints of crossbar arrays, this study adopts the *im2col* technique. While various mapping strategies exist, this approach transforms the convolution operation into General Matrix Multiplication (GEMM) [2], facilitating the direct mapping of weights onto the simulated CIM hardware.

- **Weight Unrolling:** The 4D weight tensors from convolutional layers are flattened into 2D matrices, which are subsequently programmed into the conductance states of the memory crossbar.
- **Bit-Slicing:** To accommodate the limited precision of individual analog memory cells (e.g., 1-bit or 2-bit), high-precision weights are decomposed via bit-slicing and distributed across multiple physical cells or arrays to preserve representational accuracy [9].

The mapping strategy in Figure 2b aligns with the core principles of representative designs such as ISAAC [9] and PRIME [3], which have reported superior computational throughput and energy efficiency over traditional digital baselines.

3 Algorithm Analysis and Mapping Strategy

A high-level functional modeling strategy is adopted to map the pre-trained VGG16 model onto the CIM simulation framework. By incorporating fixed-point quantization errors, array tiling, and analog circuit noise within the PyTorch environment, this approach effectively estimates the behavioral characteristics of the CIM hardware.

3.1 Quantization Strategy

To simulate the limited computational precision of physical CIM units (e.g., Int8 or Int4), a dynamic symmetric quantization scheme is implemented (via the `quantize_tensor` function). This scheme maps 32-bit floating-point weights and activations into the signed integer domain.

3.1.1 Dynamic Range Matching

For a given input tensor x, the scaling factor S is determined by its maximum absolute value $|x|_{\max}$, and the target bit-width B:

$$S = \frac{|x|_{\max}}{2^{B-1} - 1} \tag{2}$$

where B represents the quantization precision (e.g., $B = 8$).

3.1.2 Fixed-Point Mapping and Clamping

Floating-point values are scaled by S, rounded to integers, and subsequently clamped to the range $[-2^{B-1}, 2^{B-1} - 1]$ to model the storage constraints of hardware memory cells:

$$x_q = \text{clamp}\left(\text{round}\left(\frac{x}{S}\right), -2^{B-1}, 2^{B-1} - 1\right) \tag{3}$$

Negative weights utilize standard signed integer representation. This abstraction omits the complexity of modeling physical differential pair structures. While this simplifies the simulation, it assumes ideal signed arithmetic representation.

3.2 Array Mapping and Temporal Tiling

Neural network weight matrices ($D_{out} \times D_{in}$) typically exceed the physical dimensions of CIM crossbar arrays ($H_{array} \times W_{array}$, e.g., 128×128). To address this, a **Temporal Tiling** strategy is employed to map large matrices onto smaller physical arrays (in the `CIM_Tile_Sim` function).

3.2.1 Virtual Array Partitioning

The large weight matrix is logically partitioned into multiple sub-blocks (tiles). The required number of horizontal tiles N_h and vertical tiles N_v are calculated as follows:

$$N_h = \left\lceil \frac{D_{in}}{W_{array}} \right\rceil, \quad N_v = \left\lceil \frac{D_{out}}{H_{array}} \right\rceil \tag{4}$$

3.2.2 Latency Estimation Model

To establish a deterministic benchmark for mapping efficiency, the latency estimation in this study adopts a serialized execution model. Consistent with a resource-constrained hardware design scenario where a single CIM array is time-multiplexed, the processing latency is defined as the product of the input volume and the total count of mapping tiles:

$$Latency = (Batch \times Length) \times (N_h \times N_v) \tag{5}$$

As detailed in Appendix A, this metric quantifies the mapping overhead, serving as a simplified baseline for comparing different array sizes, distinct from fully parallelized hardware implementations. Furthermore, to evaluate robustness, Gaussian noise is injected at the MVM output stage ($Y_{noisy} = Y + \mathcal{N}(0, \sigma)$) to simulate thermal noise and process variations inherent in analog circuits.

3.3 Deployment Workflow

Figure 3 illustrates the end-to-end data processing workflow implemented in the `CIM_Conv2d_Sim` module. To align with the parallel computing paradigm of CIM arrays, the `im2col` technique is employed to recast convolution operations into MVM.

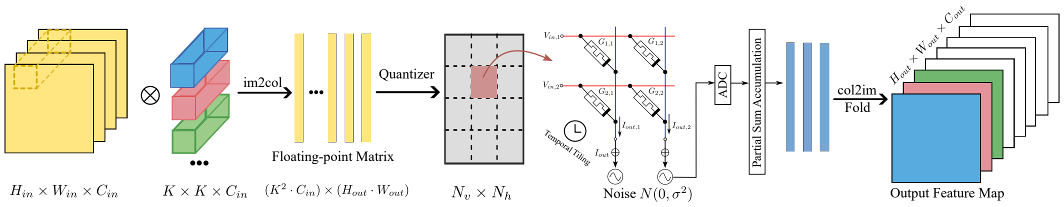

Figure 3: CIM convolution dataflow using im2col with quantization and noise simulation.

Specifically, the input data is first expanded via the `Unfold` operation in the digital domain and then quantized for analog computing simulation. Upon the completion of matrix multiplication, the results are digitized by a simulated ADC (incorporating de-quantization and re-sampling) before being reshaped into the output feature map for the subsequent layer.

4 Implementation Details

The implementation integrates the VGG16 model into the CIM simulation framework using a hybrid architecture. Computationally intensive operations, specifically convolutional and fully connected layers, are offloaded to analog CIM arrays. Conversely, lightweight non-linear functions, such as activation and pooling, remain in the digital domain to maximize energy efficiency.

4.1 Modeling Matrix-Vector Multiplication

The `CIM_Tile_Sim` class encapsulates the core behavior of the CIM array, modeling the complete signal chain from digital input quantization to analog dot-product computation and ADC readout.

To capture the non-ideal characteristics of analog computing, a dynamic noise injection mechanism, integrated within the experimental simulation framework, models stochastic behavior during inference. The operational pipeline proceeds as follows:

- **Quantized Computation:** Performs ideal matrix multiplication using quantized weights W_q and inputs X_q.
- **Noise Injection:** Simulates analog non-idealities by introducing Gaussian noise to model current fluctuations within the array.

- **ADC Readout:** Re-applies quantization to the analog output to reflect the precision limits of the Analog-to-Digital Converter.

Listing 1: Forward Propagation of the CIM Tile

```python
def forward(self, x_vector):
   # 1. Hardware Quantization (DAC & Weights)
   x_q = quantize_tensor(x_vector, CONFIG.activation_bits)
   w_q = quantize_tensor(self.weight, CONFIG.weight_bits)

   # 2. Analog MVM Execution
   output = F.linear(x_q, w_q, self.bias)

   # 3. Circuit Noise Injection
   if hasattr(CONFIG, 'noise_sigma') and CONFIG.noise_sigma > 0:
      noise = torch.randn_like(output) * CONFIG.noise_sigma
      output = output + noise

   # 4. Output Quantization (ADC Simulation)
   output_q = quantize_tensor(output, CONFIG.activation_bits)
   return output_q
```

4.2 Digital and Fully Connected Layer Implementation

Beyond standard convolution, the system maps operators to distinct hardware domains based on their computational density and data flow characteristics.

4.2.1 Digital Logic Domain

Non-linear operations, specifically ReLU activation and Max Pooling, yield minimal performance gains from in-memory processing. Consequently, these are deployed in Near-Memory Digital Logic Units. The implementation classes, `Digital_ReLU` and `Digital_MaxPool2d`, explicitly isolate digital energy consumption (`digital_ops`) from the analog energy budget.

4.2.2 Fully Connected Layers

The fully connected layers within the VGG16 classifier are structurally treated as degenerate 1×1 convolutions. The `CIM_Linear_Sim` module inherits the architecture of `CIM_Tile_Sim`, ensuring that FC weights are mapped to the CIM crossbar arrays for vector-matrix acceleration, rather than relying on general-purpose CPU or GPU computation.

4.3 Model Architecture Integration

The standard **VGG16** network serves as the benchmark for this design. A custom wrapper class, `VGG16_Sim`, automatically traverses the original PyTorch model hierarchy, replacing standard layers with their corresponding hardware-aware simulation modules.

Table 1: Mapping of VGG16 Layers to Hardware Simulation Modules

Original Layer	**Simulation Module**	**Execution Domain**
`nn.Conv2d`	`CIM_Conv2d_Sim`	**CIM Array (Analog)**
`nn.Linear`	`CIM_Linear_Sim`	**CIM Array (Analog)**
`nn.ReLU`	`Digital_ReLU`	Digital Logic
`nn.MaxPool2d`	`Digital_MaxPool2d`	Digital Logic
`nn.Dropout`	`Digital_Dropout`	Passthrough (Inference)

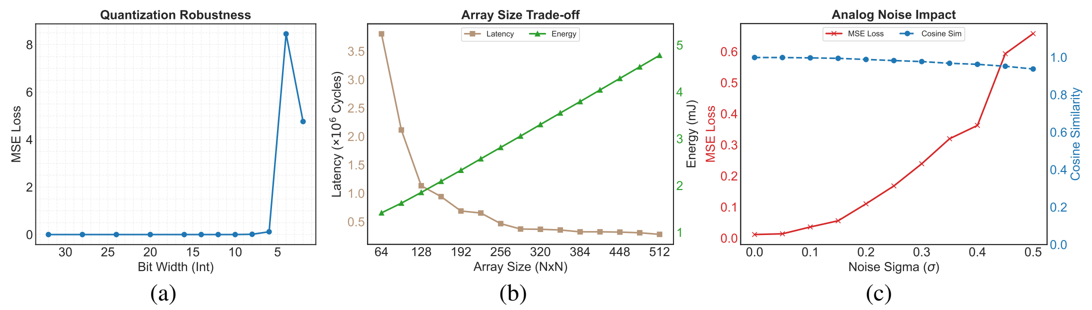

Figure 4: Design Space Exploration: (a) MSE loss analysis across different quantization bit-widths, (b) Impact of CIM array size on latency and energy consumption, and (c) Robustness analysis showing the impact of varying Gaussian noise levels on both numerical precision (MSE Loss) and feature directional stability (Cosine Similarity).

5 Experiments and Results

The performance of the proposed architecture is evaluated through three key aspects: quantization precision, array size trade-offs, and robustness to analog noise. The MSE is adopted as the primary metric to quantify the numerical deviation from the baseline floating-point implementation, while Cosine Similarity is utilized to assess the stability of feature directional alignment, which is critical for classification semantics.

5.1 Quantization Ablation Study

The quantization bit-width was adjusted from 32-bit down to 2-bit to determine the minimum precision required for reliable inference.

As illustrated in Fig 4a, high tolerance to reduced precision is observed. The MSE loss remains below 10^{-3} for bit-widths greater than 10-bit. At 8-bit precision, a minor MSE increase to **0.012** is recorded, which is generally acceptable for deployment. However, reducing the precision to 6-bit results in a noticeable error increase (MSE ≈ 0.12), followed by a drastic degradation at 4-bit quantization where the MSE peaks at **8.45**.

Notably, an anomaly is observed at 2-bit precision, where the MSE drops to **4.76**. This reduction is likely attributed to the extremely limited dynamic range of 2-bit representation, which effectively bounds the maximum magnitude of numerical errors compared to the volatile fluctuations seen at 4-bit. Despite this numerical decrease in MSE, the functional correctness is lost (Match Baseline: No) for both 4-bit and 2-bit configurations. Consequently, 8-bit quantization is identified as the optimal configuration to balance memory compression and accuracy.

5.2 Trade-off between Array Size, Latency, and Energy

The CIM array size was varied from 64×64 to 512×512 to analyze the impact on inference latency (cycles) and total energy consumption (mJ). Detailed modeling methodologies regarding the physics-aware energy estimation and latency calculation are provided in Appendix A.

The analysis in Fig 4b highlights an architectural trade-off observed under the proposed parasitic capacitance model. Larger arrays (up to 512×512) leverage higher spatial mapping density to lower the projected inference latency. However, within this simulation framework, total energy consumption rises monotonically to **4.780 mJ**. This trend is primarily driven by the modeled parasitic capacitance on longer bitlines, which, in this first-order approximation (detailed in Appenidx A), outweighs the energy savings from reduced partial sum accumulation. Consequently, smaller arrays (e.g., 64×64) appear more favorable for energy-sensitive deployments in this specific context, subject to the static power and ADC overheads of the physical implementation.

5.3 Robustness to Analog Noise

To simulate the non-ideal characteristics of analog computing, Gaussian noise with varying standard deviations ($\sigma \in [0, 0.5]$) was injected into the computation results.

Fig 4c demonstrates the robustness of the VGG16 model on the CIM architecture. While the MSE loss increases with noise intensity, rising from **0.0117** at the baseline to **0.7136** at $\sigma = 0.5$, the Cosine Similarity exhibits stability, maintaining a high value above **0.94** throughout the experiment (decreasing only marginally from **0.9988** to **0.9475**).

This indicates that although analog noise significantly distorts the absolute magnitude of activation values (high MSE), it preserves the semantic orientation of the feature vectors in the high-dimensional space. Consequently, the functional performance remains stable. Experimental logs indicate that the inference results match the noise-free baseline ("Match Baseline: Yes") even at the highest tested noise level ($\sigma = 0.5$). This implies that the decision boundaries of the over-parameterized VGG16 model are sufficiently robust to maintain correct classification as long as the feature directionality is preserved. It is worth noting that while functional accuracy is preserved here, the impact of noise may vary for more compact network architectures.

6 Conclusion and Future Work

6.1 Conclusion

This report investigated the deployment of the VGG16 architecture onto a simulated CIM platform. By mapping MVM operations directly to memory arrays, the limitations imposed by the traditional Von Neumann architecture can be significantly mitigated.

Key findings and insights include:

- **Architectural Efficiency:** Simulation results suggest that the CIM paradigm offers a potential solution to the “Memory Wall,” effectively mitigating the latency and energy bottlenecks caused by the physical separation of processing and memory units.
- **Algorithmic Robustness:** The VGG16 model exhibits significant resilience against analog noise. Notably, while the MSE increased from a baseline of 0.0117 to 0.7136 at a noise intensity of $\sigma = 0.5$, the inference outcomes remained consistent with the noise-free baseline
- **Decision Boundary Stability:** The disconnect between rising numerical error (MSE) and stable classification accuracy suggests that the model’s decision boundaries are sufficiently robust to tolerate the stochastic deviations inherent in analog computation.

6.2 Limitations and Future Work

While the experimental results demonstrate the functional robustness of the model, several limitations in the current scope must be acknowledged:

- **Simulation Environment:** The current study relies on a simulated environment to model non-ideal factors such as quantization and thermal noise. This abstraction may not fully capture the complex, non-linear behaviors of physical hardware devices.
- **MSE Divergence:** While functional accuracy was preserved up to $\sigma = 0.5$, the corresponding rise in MSE implies substantial deviations in internal feature representations. This trend points toward a potential stability threshold, suggesting that noise levels exceeding this range might eventually compromise model performance.

Future work should focus on extending the noise analysis beyond $\sigma = 0.5$ to determine the critical failure threshold of the model and validating these simulated findings on physical CIM hardware prototypes.

Acknowledgments

This report marks a significant milestone in my undergraduate studies. I would like to extend my sincere gratitude to the course instructor, Prof. Yixin Zhu. His comprehensive guidance on academic writing and scientific visualization provided me with an essential toolkit. As a novice researcher, I take full responsibility for any roughness that remains in this manuscript, which persists despite his excellent instruction.

I am deeply indebted to Prof. Xiyuan Tang and my mentor, Mr. Haikang Diao. Haikang patiently guided me through the fundamentals of CIM, welcoming me into this fascinating field. My gratitude extends equally to Prof. Tang. While our initial office hour interactions were helpful, witnessing his rigorous and meticulous approach during group meetings was truly transformative. It offered me a genuine glimpse into the spirit of serious scientific inquiry.

Presenting my preliminary work amidst discussions on cutting-edge research was initially daunting. However, both Prof. Tang and Haikang treated my efforts with immense patience and encouragement, creating an environment where I felt supported rather than judged. It has been a distinct privilege to begin my research journey under such mentorship. I feel truly fortunate to have crossed paths with Prof. Tang, Prof. Zhu, and Haikang.

References

[1] John W. Backus. Can programming be liberated from the von neumann style? a functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, 1978. 1

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning, 2014. arXiv preprint arXiv:1410.0759. 3

[3] Ping Chi, Shuangchen Li, Tao Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pages 27–39. IEEE, 2016. 2, 3

[4] Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 10–14. IEEE, 2014. 1, 2

[5] Yiqi Jing, Meng Wu, Jiaqi Zhou, Yiyang Sun, Yufei Ma, Ru Huang, Tianyu Jia, and Le Ye. AIG-CIM: A scalable chiplet module with tri-gear heterogeneous compute-in-memory for diffusion acceleration. In *Proceedings of the 61st ACM/IEEE Design Automation Conference, DAC*, pages 145:1–145:6, 2024. 2

[6] Manuel Le Gallo, Abu Sebastian, Roland Mathis, Matteo Manica, Heiner Giefers, Tomas Tuma, Costas Bekas, Alessandro Curioni, and Evangelos Eleftheriou. Mixed-precision in-memory computing. *Nature Electronics*, 1(4):246–253, 2018. 2

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. 1

[8] Abu Sebastian, Manuel Le Gallo, Reem Khaddam-Aljameh, and Evangelos Eleftheriou. Memory devices and applications for in-memory computing. *Nature Nanotechnology*, 15(7):529–544, 2020. 2

[9] Ali Shafiee, Anirudh Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pages 14–26. IEEE, 2016. 2, 3

[10] Wm A Wulf and Sally A McKee. Hitting the memory wall: implications of the obvious. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, 1995. 1

A Simulation Methodology and Energy Modeling

To evaluate the performance of the VGG16 model on the CIM architecture, a behavioral-level simulation framework was developed. This framework estimates inference latency and energy consumption based on mapping strategies and physical array characteristics. While this approach simplifies certain circuit-level details, it provides a first-order approximation of the architectural trade-offs between spatial mapping density and parasitic overheads.

A.1 Latency Estimation Model

To estimate the inference latency (L_{total}), the simulation adopts a serialized execution model. This approach is designed to isolate the impact of array sizing on mapping efficiency from specific parallelization schemes, treating the total latency as the cumulative sum of operational cycles across all layers:

$$L_{total} = \sum_{l=1}^{L} (N_{in}^{(l)} \times M_{tile}^{(l)}) \tag{6}$$

where L denotes the total number of layers, $N_{in}^{(l)}$ represents the input volume (defined as Batch $\times$ Sequence Length) for layer l, and $M_{tile}^{(l)}$ indicates the number of CIM tiles required for weight mapping. The tile count M_{tile} is determined by the physical constraints of the array dimensions ($H_{array} \times W_{array}$):

$$M_{tile} = \lceil \frac{D_{in}}{W_{array}} \rceil \times \lceil \frac{D_{out}}{H_{array}} \rceil \tag{7}$$

where D_{in} and D_{out} correspond to the input and output feature dimensions, respectively.

The serialized execution model is adopted to establish a simplified baseline for comparing mapping efficiency across different array dimensions ($H_{array} \times W_{array}$). While this approach isolates the geometric impact of array sizing from complex scheduling strategies, it represents a conservative estimate that abstracts away the multi-tile parallelism and pipelining throughput typical of physical CIM implementations.

A.2 Physics-Aware Energy Model

Total energy consumption E_{total} is modeled as the sum of dynamic computation energy (MAC), partial sum accumulation costs, and peripheral circuit overheads.

A.2.1 Intra-Array MAC Energy (E_{MAC})

To capture the primary physical impact of array scaling, a linear parasitic capacitance model is utilized. The energy per MAC operation is modeled to scale with array height, reflecting the dominant impact of bitline length on capacitive load in the absence of advanced interconnect optimizations:

$$E_{MAC}^{op} = E_{base} + H_{array} \cdot \alpha_{cap} \tag{8}$$

$$E_{MAC}^{total} = N_{MAC} \times E_{MAC}^{op} \tag{9}$$

where E_{base} is the intrinsic energy of the unit cell (0.05 pJ), and α_{cap} (0.0005 pJ/row) represents the scaling factor for parasitic capacitance. While second-order effects such as leakage variation and non-linear ADC scaling are simplified, this linear approximation serves to identify the energy penalty trends associated with increasing array dimensions.

A.2.2 Inter-Tile Accumulation Energy (E_{accum})

When the input dimension D_{in} exceeds the array width W_{array}, weights must be distributed across multiple arrays. This requires inter-tile partial sum accumulation, which incurs significant digital energy costs (SRAM read/write and arithmetic operations):

$$E_{accum}^{total} = N_{out}^{pts} \times (\lceil \frac{D_{in}}{W_{array}} \rceil - 1) \times E_{psum} \tag{10}$$

where N_{out}^{pts} is the total number of output points and E_{psum} is the energy cost per partial sum accumulation (0.5 pJ).

A.2.3 Summary of Limitations

This simulation framework focuses on high-level architectural design space exploration. It captures the fundamental trade-off: larger arrays increase E_{MAC}^{op} due to bitline capacitance, whereas smaller arrays increase E_{accum}^{total} due to data movement. However, the absolute numerical values should be interpreted as relative indicators rather than precise circuit-level predictions, as the model assumes fixed ADC energy costs and simplifies the non-linear relationship between precision and power.