

# Fast and Energy-Efficient Support for Low-Precision LLMs on PIM

Byeori Kim<sup>1</sup>, Sangjun Lee<sup>2</sup>, Eunhyeok Park<sup>2</sup>

<sup>1</sup>Department of Computer Science and Engineering, <sup>2</sup>Graduate School of Artificial Intelligence  
Pohang University of Science and Technology (POSTECH), Pohang, South Korea  
{byeori, leesangjun, eh.park}@postech.ac.kr

**Abstract**—Processing-in-Memory (PIM) has gained momentum as a promising approach for mitigating memory bottlenecks, and it is particularly well suited to autoregressive decoding in large language models (LLMs), where memory-bound General Matrix-Vector Multiplication (GEMV) operations account for a large portion of the workload. Due to the large size of LLMs, it is challenging to support them in PIM environments with limited memory capacity. Applying group-wise weight-only quantization (GWQ), widely used in LLMs, can effectively reduce model size while minimizing accuracy degradation. However, the weights in GWQ-applied LLMs are typically dequantized using scales and zero-points before GEMV is performed, which can introduce non-trivial latency overhead. In this paper, we propose a method for DRAM-PIM to efficiently support GEMV operations in symmetric and asymmetric GWQ-based LLMs with INT2 and INT4 precision. Based on the Newton scheme, the proposed method incurs an area overhead of 20.6% compared to the PIM units used in a 16-bank, single-channel system. When asymmetric GWQ with a group size of 128 is applied, it achieves approximately a 4× reduction in storage at INT4, along with a 1.16× speedup and 1.41× energy efficiency compared to FP16 GEMV. At INT2, it achieves an 8× reduction in storage, along with a 1.27× speedup and 1.57× energy efficiency.

**Index Terms**—large language model (LLM), group-wise weight-only quantization (GWQ), dequantization, general matrix vector multiplication (GEMV), processing-in-memory (PIM), DRAM

## I. INTRODUCTION

Processing-in-memory (PIM) has emerged as a promising hardware paradigm for efficient execution of large language models (LLMs). By exploiting its inherently high internal bandwidth, PIM alleviates the severe memory bottleneck that arises during autoregressive generation. Numerous designs have been proposed to leverage this advantage [1]–[8]. This benefit is especially important for mobile devices, where strict energy constraints and limited parallelism make the memory bottleneck even more challenging to overcome.

Despite this advantage, PIM faces a critical trade-off in memory capacity. Embedding compute logic and temporary storage into memory reduces the available space for data, and this problem is further aggravated by the poor area scalability of DRAM technology [1], [2]. Since memory capacity is a highly valuable resource in embedded environments, mitigating the storage overhead of in-memory computation is essential for practical deployment of LLMs.

To address this capacity challenge, group-wise weight-only quantization (GWQ) [9]–[11] has recently attracted attention. While its ability to relieve the memory wall is less relevant for PIM, GWQ’s strong compression capability remains highly

beneficial. To date, a single study [12] has extended HBM-PIM [2] to support 4-bit symmetric GWQ (SYM-GWQ), achieving a 75% reduction in weight storage with negligible quality loss. However, this approach is restricted to a fixed 4-bit SYM-GWQ, offering limited flexibility in balancing compression ratio and model accuracy.

In this paper, we present a more general and versatile PIM architecture that achieves higher compression with lower quantization error. Our design supports 2-bit and 4-bit precisions under both SYM-GWQ and asymmetric GWQ (ASYM-GWQ), enabling efficient memory use while preserving model quality. To realize this, we extend the Newton [4] PIM scheme with lightweight modifications that reconstruct low-precision weights to high-precision values on the fly. This allows existing compute units to process asymmetric quantized weights with minimal changes. Overall, our hybrid design combines the efficiency of quantization with the flexibility of full-precision models, achieving higher speed and energy efficiency compared to FP16 General Matrix-Vector Multiplication (GEMV) baseline on PIM.

## II. BACKGROUND AND RELATED WORKS

### A. LLM group-wise weight-only quantization (GWQ)

Weight-only quantization is a compression technique designed to reduce the storage overhead of LLMs by mapping their weights into 4-bit or even lower bit-width representations. Following the initial demonstration of quantization feasibility by OPTQ [9], a series of subsequent studies have explored more advanced approaches. Among them, AWQ [10] introduced group-wise weight-only quantization (GWQ), which effectively mitigates accuracy degradation and has since become a widely adopted baseline in subsequent studies [13]–[17].

In GWQ, given a weight matrix  $W \in \mathbb{R}^{O \times I}$ , where  $O$  and  $I$  represent the number of output and input channels respectively, the weights are partitioned into groups along the input dimension. For a group size  $g$ , the matrix is divided into  $O \times (I/g)$  groups. By assigning individual quantization parameters to each group, GWQ enables finer-grained representations, significantly improving post-quantization accuracy with only a modest increase in metadata overhead. Note that in some cases,  $z$  is fixed to 0 for simplicity, a setting known as SYM-GWQ [18]. While this setting simplifies implementation, it often results in lower accuracy. In contrast, ASYM-GWQ, where  $z$  can be non-zero, offers higher fidelity, particularly when targeting ultra-low-precision formats such as sub-4-bit

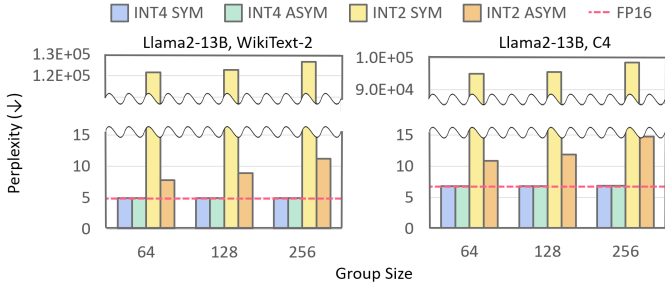


Fig. 1. Perplexity of Llama2-13B using AWQ-based INT4/INT2 weight-only quantization (symmetric and asymmetric) across different group sizes on WikiText-2 and C4.

quantization [14], [19], for which preserving output quality is more challenging.

### B. PIM for AI Acceleration

Many PIM architecture studies [2]–[5] have focused on accelerating memory-intensive GEMV operations for AI workloads. Among them, HBM-PIM [2] and Newton [4] are representative designs. Both assume that reading one column from DRAM yields 256 bits, corresponding to sixteen 16-bit floating-point values. To fully exploit this bandwidth, each PIM unit integrates 16 floating-point units (FPUs) for parallel execution. HBM-PIM [2] employs 16 multipliers and adders in a SIMD organization, supporting not only MAC operations but also additional instructions such as MUL, ADD, and MAD. In contrast, Newton [4] performs MAC operations using 16 multipliers with an adder tree of 16 adders, producing 16-bit partial sums. More recent studies [6]–[8] propose hybrid systems that integrate PIM with GPUs or NPUs to improve LLM serving. These designs also rely on GEMV execution within PIM units, often adopting Newton-like architectures. HBM-PIM, Newton, and these subsequent studies all employ 16-bit FPUs, but using only 16-bit FPUs is insufficient to support weight-only or group-wise quantization in LLMs.

## III. PRELIMINARY

### A. Importance of Asymmetric Quantization for Sub-4-bit GWQ

Early studies established the potential of GWQ in the 4-bit domain, while more advanced work has shown that sub-4-bit mixed-precision quantization enables a finer trade-off between storage and accuracy. For example, AMQ [14] offers a powerful automated toolchain for GWQ, searching for the optimal per-layer precision of LLMs by balancing compression ratio and model quality. This flexibility opens new opportunities for LLM deployment, such as fitting larger models within the same memory budget to improve accuracy or adopting more aggressively compressed models to minimize memory footprint at the cost of some quality.

Building on this advancement, we emphasize the critical role of ASYM-GWQ in enabling sub-4-bit precision. Fig. 1 presents results obtained by applying AWQ [10]-based quantization to the Llama2-13B model using various group sizes, with perplexity measured on WikiText-2 [20] and C4 [21]. We evaluated SYM-GWQ using AWQ, and evaluated ASYM-GWQ using

LLMC [19]. Since storing weights in 2-bit or 4-bit form allows more efficient data placement in DRAM subarrays, we further evaluated SYM-GWQ and ASYM-GWQ on various LLMs using AMQ [14], combining INT2 and INT4 quantization with the commonly used group size of 128. Perplexity was measured on WikiText-2 and C4, and the results are presented in Fig. 2.

The results show that around 3.5 to 4 bits of effective precision, both SYM-GWQ and ASYM-GWQ can maintain accuracy with only minor degradation. However, below 3 bits, SYM-GWQ suffers severe performance loss. This gap arises because SYM-GWQ cannot exploit the stabilizing effect of zero-points, which ASYM-GWQ effectively provides. Allowing such offsets in hardware would therefore unlock a significantly better performance–cost trade-off for sub-4-bit quantization.

### B. Scale Cascading for HBM-PIM

Performing GEMV operations on group-wise quantized LLMs requires dequantizing the quantized weights. A quantized weight  $w_{int}$  is reconstructed into a floating-point value  $w_{fp}$  using  $w_{fp} = s(w_{int} + z)$ , where  $s$  is the scale and  $z$  the zero-point. This dequantization adds one addition and one multiplication per weight, practically doubling the FLOPs compared to FP16 GEMV. On GPUs, this overhead is largely hidden due to compute underutilization caused by memory bottlenecks. In contrast, the overhead becomes significant on PIM. Here, compute capability closely matches internal bandwidth. To mitigate this, prior work [12] extended HBM-PIM [2] to support dequantization with minimal hardware and latency cost.

A key idea of this study is to postpone dequantization until after group-wise accumulation by ignoring the zero-point. In this approach, a simple precision converter (INT2FP Converter) applies a fixed scale coefficient  $s' = 1/2048$  to map INT4 data into the FP16 domain. The converted data is then processed by existing FP16 GEMV units, with the group-wise scale applied afterward. This technique, known as Scale Cascading, replaces per-weight scale multiplication with a single MUL per group, enabling efficient support for GWQ.

To further improve efficiency, a Bit Selector was added. It reduces energy consumption by fetching only the necessary lower-bit data from the row buffer for INT4 parallel processing across the 16 floating-point units. The selected data is passed through the INT2FP Converter, producing FP16 values equivalent to the scaled integer weights. Together, these techniques demonstrate the feasibility of extending existing PIM architectures to support quantized LLMs with low overhead, though the design is restricted to 4-bit SYM-GWQ on specific hardware.

## IV. SCALE CASCADING+

Quantized LLMs are generally stored in memory as integer values along with associated scales and zero-points. During inference, these values are commonly dequantized to recover FP16 weights before performing MAC operations. A straightforward implementation of GWQ in PIM would involve (1) converting integers to FP16 values, (2) applying the scale and zero-point, and (3) performing FP16 GEMV. However, this adds one conversion and one additional MUL and ADD per weight, severely degrading performance.

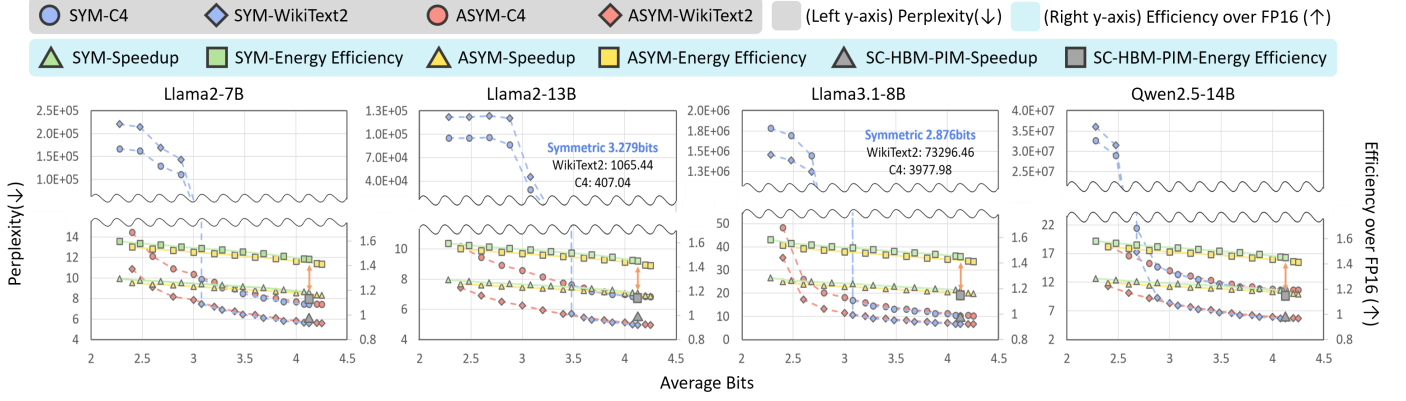


Fig. 2. Perplexity on WikiText-2 and C4 for various LLMs using symmetric and asymmetric AWQ-based INT2/INT4 layer-wise mixed-precision bit configurations discovered by AMQ. For the AMQ mixed-precision search, we used the default AMQ settings, except that the search space was limited to 2-bit and 4-bit precisions. The average bit-width includes quantization parameters such as scale and zero-point. Based on the per-layer quantized bit-widths for each model, we approximated the speedups and energy efficiency of our method for GEMV in a PIM environment. SC-HBM-PIM refers to the design proposed in prior work.

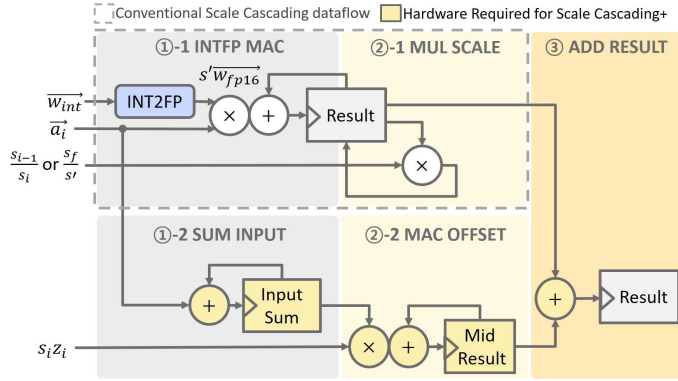


Fig. 3. Dataflow and stage pipelining of the proposed Scale Cascading+ scheme; the gray dashed box indicates the original Scale Cascading dataflow introduced in prior work.

To address this, we propose Scale Cascading+, an efficient method that exploits the key property of GWQ: all weights in a group share the same scale and zero-point. Building on the original Scale Cascading [12], which supports only scales without zero-point, we extend the concept to support a broader range of quantization formats, including symmetric and asymmetric quantization. The core idea of Scale Cascading+ is to decouple dequantization from computation via reordering. Let  $\vec{w}_i$  and  $\vec{a}_i$  be the quantized weights and activations in group  $i$ , and let  $s_i$  and  $z_i$  be the corresponding scale and zero-point. The original output based on a naive dequantization scheme that performs dequantization for each weight is given as follows:

$$y = \sum_i (\vec{w}_i + z_i) s_i \cdot \vec{a}_i \quad (1)$$

Scale Cascading+ modifies the original Scale Cascading so that the handling of scales can be performed independently of zero-point processing. Scale Cascading+ is formulated as:

$$\begin{aligned} y_0 &= s' \vec{w}_0 \cdot \vec{a}_0, & \alpha_0 &= s_0 z_0 S(\vec{a}_0) \\ y_i &= s' \vec{w}_i \cdot \vec{a}_i + \frac{s_{i-1}}{s_i} y_{i-1}, & \alpha_i &= s_i z_i S(\vec{a}_i) \end{aligned} \quad (2)$$

For each group, the integer weights are converted into FP16 values corresponding to the product with the shared scale  $s'$ . These values are then multiplied with the activations. The results are accumulated recursively, adjusting for inter-group scale differences. In addition, for group  $i$ , the required value for processing  $z_i$ , denoted by  $\alpha_0$ , is computed separately from the scale processing. Here,  $S(\cdot)$  denotes the sum over all elements of a vector. Finally, we apply the original scale and offset corrections to recover the accurate result as follows:

$$y = \frac{s_f}{s'} y_f + \sum_i \alpha_i \quad (3)$$

This method offers several key advantages, particularly tailored to DRAM-based PIM: 1) Low-cost conversion: By using a shared scale  $s'$ , the simple boolean logic can efficiently convert integers to FP16. 2) Overflow prevention: The use of  $s'$  ensures that accumulations remain within the precision limits of the buffer. 3) Robust scaling: Recursive rescaling between groups via  $s_{i-1}/s_i$  allows safe accumulation even when scales differ significantly. 4) Accurate final recovery: The final step ensures that original scale and zero-point values are fully restored, preserving numerical correctness. Scale Cascading+ thus enables efficient dequantization and GEMV execution within PIM architectures.

Fig. 3 illustrates the dataflow of Scale Cascading+ in hardware. For symmetric quantization, only the operations within the dashed box are required. When using asymmetric quantization with zero-points, three additional stages are introduced: 1) SUM INPUT – accumulates activations for each group, 2) MAC OFFSET – computes the offset term  $s_i z_i \cdot S(\vec{a}_i)$ , and 3) ADD RESULT – adds the offset term to the partial result, which already has the cascaded scales applied. In the diagram, boxes with the same color indicate operations that can be executed in parallel, helping to minimize overall latency.

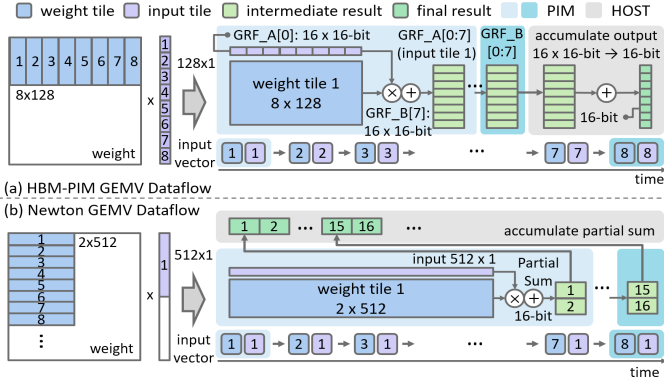


Fig. 4. Baseline GEMV dataflows for the two designs: (a) GEMV dataflow in HBM-PIM and (b) GEMV dataflow in Newton. The figure illustrates how GEMV proceeds over 1,024 weights. The numbers 1 through 8 on each weight tile and input tile indicate the processing order.

## V. PIM ARCHITECTURE FOR SCALE CASCADING+

### A. Motivation of the PIM Architecture

Prior work [12] reported overall performance degradation relative to FP16 GEMV operations. To achieve speedup, we compared and analyzed the GEMV data mapping and dataflow of HBM-PIM and Newton, another representative PIM architecture. From this analysis, our observation is that the Newton-based design is able to support Scale Cascading+ with higher speed and energy efficiency.

Fig. 4 shows the GEMV data mapping and dataflow of HBM-PIM and Newton. As shown in Fig. 4(a), each PIM unit loads up to 128 FP16 input elements into its 8 GRF\_A registers and accumulates the final outputs in 8 GRF\_B registers. This approach avoids intermediate communication between the host and DRAM, but requires repeated loading of the same inputs, leading to some overhead. On the other hand, Newton’s dataflow (Fig. 4(b)) stores 512 FP16 input elements in a global buffer located near the peripheral logic. These inputs are reused across all weight computations without reloading. However, Newton outputs partial results to the host, which then aggregates them to obtain the final output.

Due to their different dataflows, the two architectures also differ in how weights are mapped to DRAM rows. In HBM-PIM, weights are mapped across two DRAM rows to form an  $8 \times 128$  tile, aligning with the accumulation across 8 GRF\_B. In contrast, Newton stores a  $1 \times 512$  tile in a single DRAM row and computes partial results by multiplying these weights with the 512 input elements, sending the results back to the host for final aggregation. INT4 and INT2 data can store 2,048 and 4,096 weights per DRAM row, respectively, providing the advantage of reduced bank row activation latency during PIM operations. While the dataflow and mapping scheme of HBM-PIM make it difficult to exploit this advantage, Newton can reduce row activations more efficiently, leaving room for performance improvement. Specifically, using 512 inputs together with the weights stored in a single DRAM row enables the generation of multiple partial results.

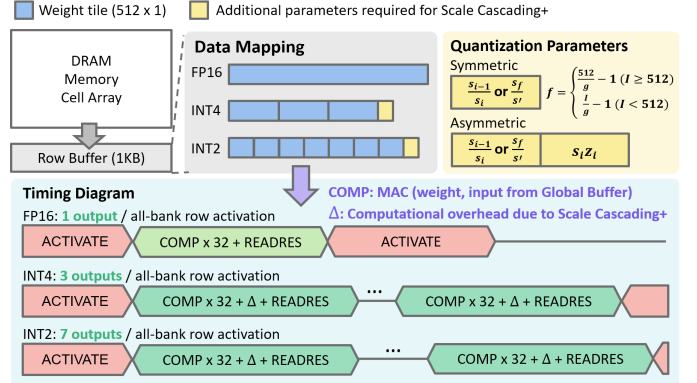


Fig. 5. Data mapping method in Newton for FP16 and for INT4/INT2 with Scale Cascading+. To perform MAC operations over 512 inputs, FP16 weights use 32 DRAM columns, INT4 weights use 8 columns, and INT2 weights use 4 columns, each producing one partial output.

### B. Newton-aware Data mapping and Scale Cascading+

For data mapping, it is necessary to consider the mapping of group-wise quantization parameters introduced by GWQ. In Newton, each bank contains one PIM unit, and activating all banks incurs a considerable delay of at least  $3 \times t_{FAW}$ . As shown in Fig. 5, to reduce DRAM activation overhead with group-wise parameters, we map the group parameters corresponding to weights processed in the same DRAM row. We consider group sizes of 64, 128, and 256 for GWQ. With INT2 ASYM-GWQ at group size 64, up to six results can be mapped in a single DRAM row, while other INT2 cases allow seven and INT4 cases allow three partial results.

Cascaded scales ( $s_{i-1}/s_i$  and  $s_f/s'$ ) used in Scale Cascading+ need to be handled considering the Newton architecture and its original dataflow. Newton reuses 512 input elements as much as possible and performs MAC operations using these inputs and the weights. When the input dimension exceeds 512, cascaded scales for Scale Cascading+ are generated by partitioning the input dimension into 512-sized segments and creating the required parameters within each segment according to the group size. This allows the dataflow to be directly applied to obtain the final result.

### C. Datapath with pipelining

As illustrated in Fig. 6, the datapath for implementing Scale Cascading+ on Newton [4] can be derived from Fig. 3. We extend Newton’s original GEMV unit with minimal additional hardware to support INT4/INT2 SYM- and ASYM-GWQ. Bit Selector+ and INT2FP Converter+ are extended from prior work [12] to support not only INT4 but also INT2. To handle the MUL SCALE stage and the MAC OFFSET stage, a 256-bit buffer is added to preload the required parameters. Newton broadcasts input elements from a global buffer near the peripheral area to all PIM units during computation. Leveraging this feature, our design performs INT2FP MAC operations while simultaneously accumulating the sum of input elements in the peripheral area. At the end of the computation, the final result is produced by adding the results of MAC OFFSET stage and MUL SCALE stage, and then transmitted to the host.

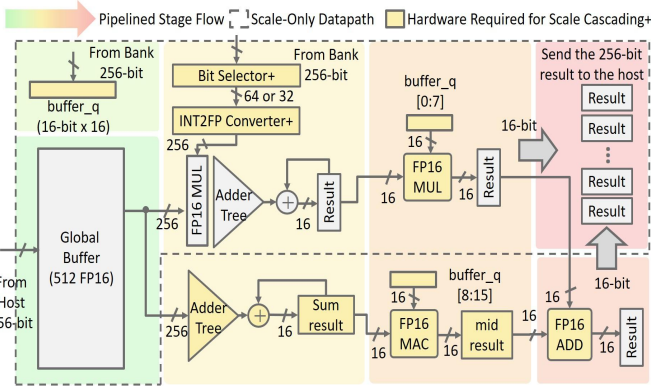


Fig. 6. Datapath of the Newton GEMV pipeline extended to support Scale Cascading+. Stages advance in the direction indicated by the top-left arrow; operations inside each colored box are processed in that stage. In the symmetric case, `buffer_q` is used only for the scales.

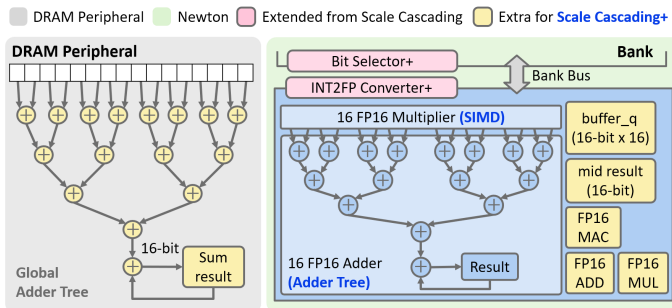


Fig. 7. Extended microarchitecture required to support Scale Cascading+.

#### D. Additional Hardware

The corresponding hardware extensions are summarized in Fig. 7. A global adder tree is placed in the DRAM peripheral area for the SUM INPUT stage illustrated in Fig. 3, and a Bit Selector+ and an INT2FP Converter+ are required near each Newton bank. Also, each PIM unit is equipped with a 256-bit `buffer_q` to store the quantization parameters ( $s_{i-1}/s_i$ ,  $s_f/s'_f$ , and  $s_i z_i$ ). During the processing of 512 input elements, group-wise intermediate results are stored in a mid result. The architecture uses FP16 MUL, MAC, and ADD units in each PIM unit to implement the MUL SCALE, MAC OFFSET, and ADD RESULT stages shown in Fig. 3.

## VI. EXPERIMENTS

### A. Experiment Setup

To evaluate the strengths of our approach, we conduct an in-depth analysis, including error analysis of Scale Cascading+ as well as evaluations of area, speedup, and energy. We measured area overhead, performance, and energy efficiency using the system configuration and timing parameters in Table I. For the Newton bank area, we calculated the area of one bank based on the values reported in [4]. While prior work [12] focused only on INT4 SYM-GWQ with HBM-PIM, Scale Cascading+ can also be supported on HBM-PIM by designing the stages as in Fig. 3, resulting in a datapath similar to Fig. 6. Since HBM-PIM does not use global I/O during PIM operations, only an

TABLE I  
SYSTEM AND DRAM TIMING SETUP FOR NEWTON AND HBM-PIM

system configuration	Newton timing		HBM-PIM timing		
channels (or pCH)	1	tRCD	14	tRCD	14
ranks	1	tCCD_L	4	tCCD_L	4
banks	16	tRAS	34	tRAS	33
columns per row	32	tRP	14	tRP	14
DRAM data block	32B	tWR	16	tWR	16
Newton Bank	32MB	CL	14	RL	20
Newton PIM units	16	tRFC	260	tRFC	350
HBM-PIM PIM units	8	tFAW	30		

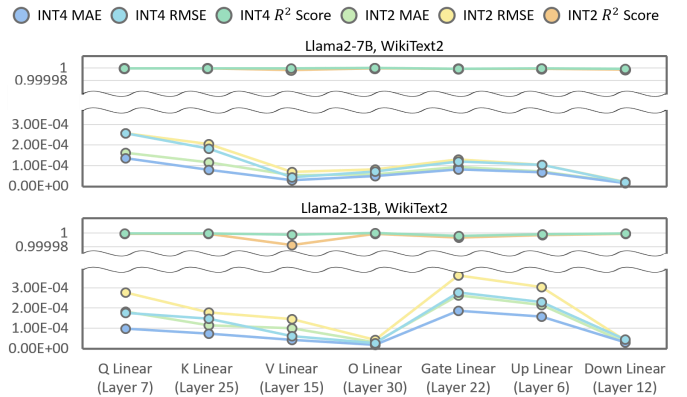


Fig. 8. MAE, RMSE, and  $R^2$  score of outputs obtained from GEMV operations using naive dequantization and Scale Cascading+. Experiments were conducted by randomly selecting linear layers from each model and applying INT2/INT4-quantized weights with asymmetric AWQ.

additional stage is required to transfer the input sum obtained externally to the PIM unit. To compare how efficiently Newton-based Scale Cascading+ can be supported, we also evaluated HBM-PIM-based Scale Cascading+ under conditions designed to maximize its efficiency.

### B. Scale Cascading+ Validation

Unlike in (1), Scale Cascading+ uses  $s'$  and postpones the processing of zero-points and scales, which may lead to differences from the final outputs obtained by (1). To examine the extent of these differences, we randomly selected linear layers from the Llama2-7B and 13B models and performed GEMV using weights quantized with INT2/INT4 ASYM-GWQ (group size = 128) with activations obtained from WikiText-2. We then measured the Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and  $R^2$  score of the results obtained by (1) and by Scale Cascading+. As shown in Fig. 8, the MAE and RMSE were below  $4 \times 10^{-4}$  in all cases, and the  $R^2$  score was close to 1 across all cases. This indicates that the computed results are nearly identical to those of the baseline method.

### C. Hardware Area Overhead

To measure area overhead, we synthesized the hardware units used in the baseline Newton and the additional hardware required to support Scale Cascading+ in Verilog RTL using OpenROAD-flow-scripts [22] with the NanGate 45nm library. The area and count of the hardware used in one channel

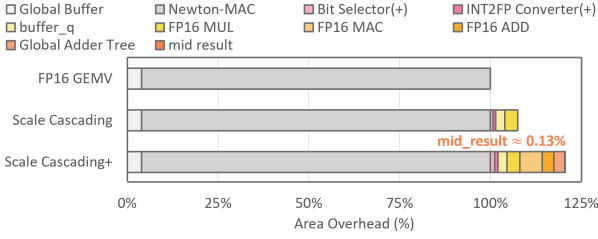


Fig. 9. Area overhead incurred by adding Scale Cascading and Scale Cascading+ support to the Newton architecture.

within a 16-bank system were evaluated, and the area overhead relative to Newton is shown in Fig. 9. When considering Scale Cascading [12] with INT4 SYM-GWQ only, Newton requires a Bit Selector, an INT2FP Converter, a 256-bit buffer\_q, and an FP16 multiplier per bank, incurring an overhead of 7.5% over the baseline Newton. When supporting Scale Cascading+, the additional hardware highlighted in pink and yellow in Fig. 7 incurs an overhead of 20.6% compared to the baseline Newton. For HBM-PIM-based systems extended to support Scale Cascading+, the minimal additional hardware includes a global adder tree, per-bank Bit Selector+, INT2FP Converter+, FP16 MAC, FP16 ADD, and buffers for storing the values required for processing in each PIM unit. Using the same calculation method as in Fig. 9, the overhead of HBM-PIM-based design was measured at approximately 15.3%, considering only the registers and FPU units used in a single channel. We also measured the area overhead of the Bit Selector+ and INT2FP Converter+ relative to the DRAM bank area. The Newton bank area was estimated using CACTI-7 [23] DDR4 22nm model to be 3.75401 mm<sup>2</sup>. The Bit Selector+ and INT2FP Converter+ add 0.074% overhead relative to a Newton bank.

#### D. Performance and Energy Efficiency

To objectively compare the speedup and energy efficiency of our method, we used two baselines: 1) FP16 GEMV and 2) an HBM-PIM-based design based on prior work [12] with Scale Cascading+ (SC+). We evaluated a Newton-based architecture and an HBM-PIM-based architecture using in-house simulators based on DRAMSim3 [24] and DRAMSim2 [25], respectively. Simulations assumed ample bank capacity. Speedups and energy efficiency were measured for GEMV kernels with INT4/INT2 SYM-GWQ (INT4 SYM SC+, INT2 SYM SC+) and ASYM-GWQ (INT4 ASYM SC+, INT2 ASYM SC+). For INT3 SYM SC+ and INT3 ASYM SC+, average cycle counts and energy consumption from INT2 and INT4 executions were used. We measured GEMV with square matrices and vectors of size  $m \in \{512, 1024, 2048, 4096, 8192\}$ , using group sizes of 64, 128, and 256. The left figures in Fig. 10 are normalized to FP16 GEMV, while the right figures are further normalized to the HBM-PIM-based design, after normalizing both to FP16 GEMV. Since similar performance and energy efficiency were observed regardless of GEMV size, we report the geometric mean across different GEMV cases for each group size. For the energy efficiency analysis, we adopted the 4× DRAM read power assumption from [4], together with the breakdown from HBM-PIM [2], since Newton lacks a detailed power

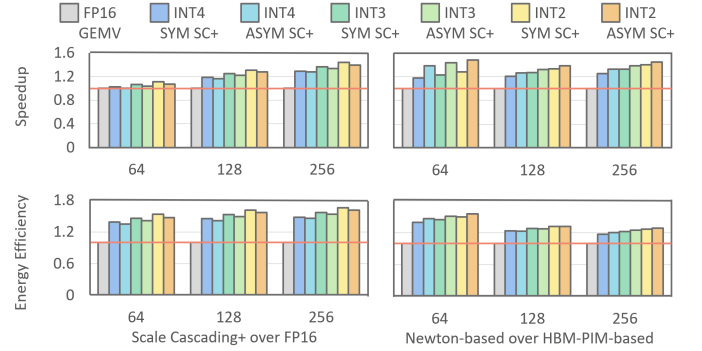


Fig. 10. Speedup and energy efficiency results vs. group size (x-axis), normalized to FP16 GEMV (left) and to HBM-PIM with Scale Cascading+ (right), using FP16 GEMV as the baseline for the right.

breakdown. The approximation was scaled with a focus on the reduction in data movement through the local bus and column decoder of the bank enabled by the Bit Selector+.

In our Newton-based method, the mapping strategy in Fig. 5 reduces DRAM row activation delay, enabling speedup and, in turn, improving energy efficiency. For INT4 ASYM SC+ with group size ( $g$ ) of 64, a slight slowdown was observed with a speedup of 0.9998×, while all other cases achieved speedups over FP16 GEMV. As  $g$  decreases, the frequency of handling scales and zero-points increases, leading to relatively lower speedups and energy efficiency. At the commonly used  $g=128$ , the speedups of INT4 (SYM SC+, ASYM SC+) and INT2 (SYM SC+, ASYM SC+) were (1.19×, 1.16×) and (1.31×, 1.27×), respectively. In terms of energy efficiency, the corresponding values were (1.45×, 1.41×) and (1.61×, 1.57×).

While [12] supported only INT4 SYM-GWQ (same as INT4 SYM SC+), our method achieved speedups and energy efficiency of 1.18× and 1.40× at  $g=64$ , 1.21× and 1.23× at  $g=128$ , and 1.25× and 1.17× at  $g=256$ , respectively. Overall, with SC+ support enabled, the Newton-based design achieved a larger speedup in ASYM SC+ at  $g=64$  than at  $g=128$  and  $g=256$ . Otherwise, speedup increased with  $g$ . In both SYM SC+ and ASYM SC+, the Newton-based design’s relative energy efficiency (vs. HBM-PIM) decreased as  $g$  increased.

## VII. CONCLUSION

We propose Scale Cascading+ and apply it to Newton, enabling PIM to support INT2/INT4 SYM- and ASYM-GWQ while maintaining compatibility with FP16 GEMV. With an efficient mapping that reduces DRAM activation delay overhead and the dataflow design of Scale Cascading+, our method achieves substantial memory savings of approximately 4× for INT4 and 8× for INT2. In most cases, it achieves higher speedup and energy efficiency over FP16 GEMV for group sizes of 64, 128, 256, with a 20.6% area overhead for Newton PIM units in a single channel.

## ACKNOWLEDGMENT

This work was supported by IITP and NRF grant funded by the Korea government(MSIT) (RS-2019-II191906, RS-2024-00396013, RS-2024-00415602, RS-2025-02218259)

## REFERENCES

- [1] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "Mcdram: Low latency and energy-efficient matrix computations in dram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.
- [2] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for pim based on commercial dram technology : Industrial product," in *International Symposium on Computer Architecture (ISCA)*, 2021.
- [3] J. H. Kim, S.-h. Kang, S. Lee, H. Kim, W. Song, Y. Ro, S. Lee, D. Wang, H. Shin, B. Phuah *et al.*, "Aquabolt-xl: Samsung hbm2-pim with in-memory processing for ml accelerators and beyond," in *2021 IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–26.
- [4] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. N. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.
- [5] Y. Kwon, K. Vladimir, N. Kim, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, G. Kim, B. An, J. Kim, J. Lee, I. Kim, J. Park, C. Park, Y. Song, B. Yang, H. Lee, S. Kim, D. Kwon, S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, M. Lee, M. Shin, M. Shin, J. Cha, C. Jung, K. Chang, C. Jeong, E. Lim, I. Park, J. Chun, and S. Hynix, "System architecture and software stack for gddr6-aim," in *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022, pp. 1–25.
- [6] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "Attacc! unleashing the power of pim for batched transformer-based generative model inference," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.
- [7] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.
- [8] M. Seo, X. T. Nguyen, S. J. Hwang, Y. Kwon, G. Kim, C. Park, I. Kim, J. Park, J. Kim, W. Shin, J. Won, H. Choi, K. Kim, D. Kwon, C. Jeong, S. Lee, Y. Choi, W. Byun, S. Baek, H.-J. Lee, and J. Kim, "Ianus: Integrated accelerator based on npu-pim unified memory system," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 545–560. [Online]. Available: <https://doi.org/10.1145/3620666.3651324>
- [9] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "OPTQ: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: <https://openreview.net/forum?id=tcbBPnfwxS>
- [10] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," in *Machine Learning and Systems*, 2024.
- [11] C. Lee, J. Jin, T. Kim, H. Kim, and E. Park, "Owq: outlier-aware weight quantization for efficient fine-tuning and inference of large language models," in *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. [Online]. Available: <https://doi.org/10.1609/aaai.v38i12.29237>
- [12] B. Kim, C. Lee, G. Kim, and E. Park, "Cost-effective extension of dram-pim for group-wise llm quantization," *IEEE Comput. Archit. Lett.*, vol. 24, no. 1, p. 53–56, Feb. 2025. [Online]. Available: <https://doi.org/10.1109/LCA.2025.3532682>
- [13] C. Lee, J.-g. Jin, Y. Cho, and E. Park, "QEFT: Quantization for efficient fine-tuning of LLMs," in *Findings of the Association for Computational Linguistics: EMNLP 2024*. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 13 823–13 837. [Online]. Available: <https://aclanthology.org/2024.findings-emnlp.811/>
- [14] S. Lee, S.-t. Woo, J.-g. Jin, C. Lee, and E. Park, "AMQ: Enabling AutoML for mixed-precision weight-only quantization of large language models," in *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Suzhou, China: Association for Computational Linguistics, Nov. 2025, pp. 35 520–35 538. [Online]. Available: <https://aclanthology.org/2025.emnlp-main.1799/>
- [15] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "Omniquant: Omnidirectionally calibrated quantization for large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=8Wuvhh0LYW>
- [16] Z. Yuan, Y. Shang, and Z. Dong, "PB-LLM: Partially binarized large language models," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=BifeBRhikU>
- [17] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: outlier-free 4-bit inference in rotated llms," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, ser. NIPS '24. Red Hook, NY, USA: Curran Associates Inc., 2024.
- [18] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," 2021. [Online]. Available: <https://arxiv.org/abs/2106.08295>
- [19] R. Gong, Y. Yong, S. Gu, Y. Huang, C. Lv, Y. Zhang, D. Tao, and X. Liu, "LLMC: Benchmarking large language model quantization with a versatile compression toolkit," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. Miami, Florida, US: Association for Computational Linguistics, Nov. 2024, pp. 132–152. [Online]. Available: <https://aclanthology.org/2024.emnlp-industry.12/>
- [20] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," in *International Conference on Learning Representations*, 2017. [Online]. Available: <https://openreview.net/forum?id=Byj72udxe>
- [21] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 1, Jan. 2020.
- [22] T. Ajayi, D. Blaauw, T. Chan, C. Cheng, V. Chhabria, D. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça *et al.*, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," *Proc. GOMACTECH*, pp. 1105–1110, 2019.
- [23] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, Jun. 2017. [Online]. Available: <https://doi.org/10.1145/3085572>
- [24] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, p. 106–109, Jul. 2020. [Online]. Available: <https://doi.org/10.1109/LCA.2020.2973991>
- [25] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "Dramsim2: A cycle accurate memory system simulator," *IEEE Computer Architecture Letters*, 2011.