

# FLARE: FP-Less PTQ and Low-ENOB ADC Based AMS-PiM for Error-Resilient, Fast, and Efficient Transformer Acceleration

Donghyeon Yi<sup>1</sup>, Seoyoung Lee<sup>1</sup>, Jongho Kim<sup>1</sup>,  
Junyoung Kim<sup>1</sup>  
KAIST<sup>1</sup>  
Republic of Korea  
{yicoreen,seoyoung,jongho,lotanda17}@kaist.ac.kr

Sohmyung Ha<sup>2\*</sup>  
NYU Abu Dhabi<sup>2</sup>  
United Arab Emirates  
sh169@nyu.edu

Ik-Joon Chang<sup>3\*</sup>  
Kyung Hee University<sup>3</sup>  
Republic of Korea  
ichang@khu.ac.kr

Minkyu Je<sup>1\*</sup>  
KAIST<sup>1</sup>  
Republic of Korea  
mkje@kaist.ac.kr

## Abstract

Encoder-based transformers, powered by self-attention layers, have revolutionized machine learning with their context-aware representations. However, their quadratic growth in computational and memory demands presents significant bottlenecks. Analog-Mixed-Signal Process-in-Memory (AMS-PiM) architectures address these challenges by enabling efficient on-chip processing. Traditionally, AMS-PiM relies on Quantization-Aware Training (QAT), which is hardware-efficient but requires extensive retraining to adapt models to AMS-PiMs, making it increasingly impractical for transformer models. Post-Training Quantization (PTQ) mitigates this training overhead but introduces significant hardware inefficiencies. PTQ relies on dequantization-quantization (DQ-Q) processes, floating-point units (FPUs), and high-ENOB (Effective Number of Bits) analog-to-digital converters (ADCs). Particularly, High-ENOB ADCs scale exponentially in area and energy ( $2^{\text{ENOB}}$ ), reduce sensing margins, and increase susceptibility to process, voltage, and temperature (PVT) variations, further compounding PTQ's challenges in AMS-PiM systems. To overcome these limitations, we propose FLARE, an AMS-PiM architecture that eliminates DQ-Q processes, introduces FPU- and division-free nonlinear processing, and employs a low-ENOB-ADC-based sparse Matrix Vector multiplication technique. Using the proposed techniques, FLARE improves error resiliency, area/energy efficiency, and computational speed while preserving numerical stability. Experimental results demonstrate that FLARE outperforms state-of-the-art GPUs and conventional PiM architectures in energy efficiency, latency, and accuracy, making it a scalable solution for the efficient deployment of transformers.

## Keywords

Self-Attention, Analog-Mixed-Signal, Process-in-Memory, PTQ, Bit-wise Sparsity, Error Resiliency, Floating-Point Unit (FPU)

## ACM Reference Format:

Donghyeon Yi<sup>1</sup>, Seoyoung Lee<sup>1</sup>, Jongho Kim<sup>1</sup>, Junyoung Kim<sup>1</sup>, Sohmyung Ha<sup>2</sup>, Ik-Joon Chang<sup>3</sup>, and Minkyu Je<sup>1</sup>. "FLARE: FP-Less PTQ and Low-ENOB

\*Corresponding Authors

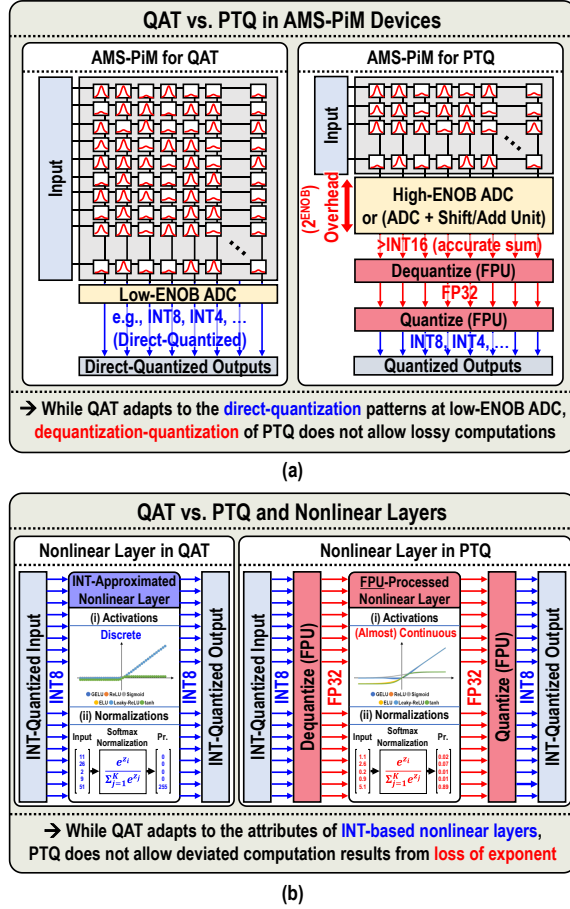
ADC Based AMS-PiM for Error-Resilient, Fast, and Efficient Transformer Acceleration," arXiv Preprint. 2024.

## 1 Introduction

Transformer models [1, 8, 16, 17, 37, 50], built on the attention mechanism [50], have become foundational in machine learning across diverse domains. While decoder-based architectures [1, 8], such as those powering chatbot services, have gained substantial popularity, encoder-based models [16, 17, 37], like BERT and Vision Transformers (ViT), remain critical for many tasks. However, it is important to note that encoder-based transformers process entire input sequences within their self-attention layers. Due to this processing, as the sequence length increases, the computational intensity and intermediate memory traffic of self-attention layers grow quadratically, posing a significant challenge in efficiently processing encoder-based transformers especially in edge devices.

Analog-Mixed-Signal Process-in-Memory (AMS-PiM) [6, 11, 24, 26, 29, 31, 35, 39, 41, 43, 45, 46, 54, 55, 57, 58] has emerged as a promising solution to address those challenges, enabling efficient execution of level-2 and level-3 BLAS operations (Basic Linear Algebra Subprograms, primarily matrix multiplications) directly within memory. These BLAS operations represent a significant portion of the self-attention layers, making AMS-PiM highly effective for reducing off-chip data movement and energy consumption. Besides, AMS-PiM's effectiveness does not extend to non-BLAS tasks, such as quantization processes or nonlinear-layer computations, requiring additional hardware and computational support. Concurrently, most AMS-PiM architectures rely on integer quantization techniques for both stationary weights (e.g., weights for Q (query), K (key), V (value), and O projections) and dynamic activations (e.g., input embeddings, Q/K/V values, attention scores, and final outputs), which are crucial for attention operations in transformers. Despite recent advancements in low-precision floating-point (FP) quantization methods [15, 53], AMS-PiM remains constrained to integer quantization due to the substantial overhead required for exponent alignment in FP arithmetic, or the significant accuracy degradation if alignment is omitted.

Quantization-Aware Training (QAT) [21, 23, 59] and Post-Training Quantization (PTQ) [5, 53] are two primary techniques for enabling quantization. QAT, widely adopted in AMS-PiM, optimizes



**Figure 1: Limitations of PTQ during inference optimizations. (a) The Dequantization-Quantization (DQ-Q) process and (b) nonlinear layers induce high-ENOB ADCs and FPUs that induce extreme area/energy overhead.**

area/energy efficiency by enabling direct quantization with low-ENOB (Effective Number of Bits) ADCs (Analog-to-Digital Converters). Through retraining, QAT allows models to adapt to characteristics of ADCs and integer-approximated nonlinear layers, eliminating the need for dequantization-quantization (DQ-Q) steps and FPUs, as visualized in the left side of Fig. 1-(a). However, QAT’s retraining process introduces substantial overhead, making it increasingly impractical for growing transformer models.

Accordingly, PTQ has become a preferred method as a retraining-free alternative to Quantization-Aware Training (QAT). However, as shown on the right side of Fig. 1-(a), PTQ introduces three major obstacles in AMS-PiM architectures as well, including (1) reliance on high-ENOB ADCs, (2) susceptibility to process, voltage, and temperature (PVT) variations, and (3) the need for Floating-Point Units (FPUs).

Typically, PTQ requires precise partial sums and scaling factors to maintain accuracy, for transformer models with large embedding depths ( $\geq 1k$ ) and high bit-precision requirements ( $\geq 4$  bits per activation and weight). These demands necessitate ADCs with ENOB of  $\geq 18$  bits to prevent quantization errors, as PTQ lacks

the adaptability of QAT to lower-ENOB ADCs via retraining. Attempting PTQ with low-ENOB ADCs is not impossible, but would result in excessive latency bottleneck due to the immense computation cycles from segmenting computations into smaller units. The main concern with the High-ENOB ADCs is that they exacerbate hardware inefficiencies, Their area and power consumption scale exponentially with  $2^{\text{ENOB}}$  [12], and the narrower sensing margins that increase with ENOB make the system more vulnerable to errors caused by PVT variations. Furthermore, PTQ’s reliance on accurate scaling factors during the DQ-Q process necessitates FPUs to handle division operations and avoid cumulative rounding errors, adding to system complexity.

Moreover, nonlinear layers for PTQs introduce additional challenges, as depicted in Fig. 1-(b). While integer-based approximations of nonlinear functions and normalization layers [32, 37] work effectively in QAT-based systems, they often lead to significant computation deviations when applied to PTQ. As a result, PTQ relies on dequantization-quantization (DQ-Q) processes and floating-point units (FPUs) to ensure accurate nonlinear-layer processing. This dependency further increases system complexity and overhead.

To address these challenges, we propose FLARE, a novel AMS-PiM architecture that holistically tackles PTQ’s inefficiencies. FLARE eliminates high-ENOB ADC, division arithmetics, and FPUs while introducing dequantization-free PTQ and integer-processed nonlinear layers. Furthermore, by leveraging low-ENOB-ADC-based sparse General Matrix Vector (GEMV) operations with bitwise sparsity, FLARE ensures fast, error-resilient, and efficient computation. With a detailed analysis of end-to-end attention computations throughout our design, we enable on-device end-to-end kernel fusion.

In summary, FLARE achieves:

- end-to-end on-chip processing of self-attention layers, reducing quadratic off-chip tensor traffic;
- INT-only, yet accurate, dequantization-free PTQ and nonlinear-layer processing, to maintain precision without high-ENOB ADC, division, or FPUs;
- fast, accurate, and efficient sparse GEMV operations with  $6\text{-}\sigma$  confidence leveraging low-ENOB ADC within MRAM-SRAM hybrid AMS-PiM arrays.

These advancements provide a scalable and energy-efficient solution for transformer inference, addressing the distinct inference-time bottlenecks of encoder-based models.

## 2 Backgrounds

### 2.1 Analog-Mixed-Signal Process-in-Memory (AMS-PiM)

Process-in-Memory (PiM) [6, 10, 18, 19, 22, 24, 26, 29, 31, 35, 36, 39, 41, 44–48, 51, 54, 55, 57, 58] architectures are highly regarded for their ability to execute level-2 (i.e., GEMV) and level-3 (i.e., GEMM) BLAS operations inherently and efficiently, which contribute to a major portion of most deep neural networks (DNNs). PiM devices allocate matrices onto memory arrays and fetch vectors (which can also form matrices) along wordlines (WLs), bitlines (BLs), or sourcelines (SLs) to execute such BLAS operations, as illustrated in

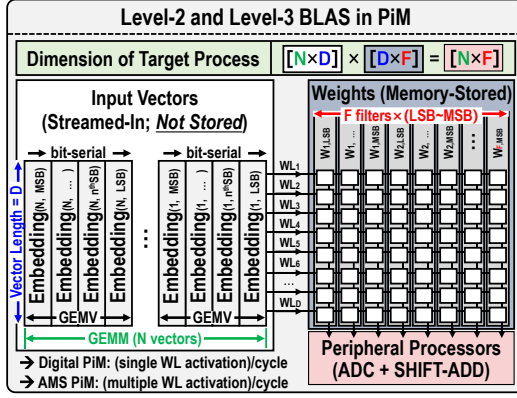


Figure 2: Design example for typical PiM devices, where the level-2 (GEMV) and level-3 (GEMM) BLAS operations are processed inherently.

Fig. 2. In the illustrated configuration, weights of  $F$  filters are stored across the memory array, inputs of depth  $D$  are fetched to  $WL_{1 \sim D}$  bit-serially for  $m$ -bit representation, and partial sums for each bit position are obtained at BLs. By integrating BLAS operations [7] directly within the memory storage, PiM drastically reduces data movement overhead and associated latency.

Analog-Mixed-Signal Process-in-Memory (AMS-PiM) [6, 10, 18, 24, 26, 31, 35, 36, 39, 44–47, 55, 57] architectures, particularly, enhance the computational efficiency over their digital counterparts. This is achieved by enabling the simultaneous activation of multiple memory interfaces, such as WLs, BLs, and SLs, allowing multiple concurrent computations within a memory array. While in digital PiM a single WL is activated per cycle to perform the column sum operations; i.e., pop (“1” or “on-cell”)–count operations, similar to a conventional read operation, contrarily, AMS-PiM simultaneously activates multiple WLs and converts the analog sum into digital signal via Analog-to-Digital Converters (ADCs), enabling multiple concurrent computations.

## 2.2 Self-Attention Layer in Transformer Encoders

For the attention mechanism in a transformer model (described in Fig. 3), each token in the input is represented as a vector of dimension  $D$ . Each sequence/image of  $B$  input batches to the self-attention layer consists of  $N$  tokens, resulting in an input tensor of dimension  $[B, N, D]$ . The input is processed through the attention mechanism with several steps, each contributing to the tensor traffic.

**1. QKV Projections:** The initial step involves projecting each input sequence into three distinct weight matrices to produce: Query (Q), Key (K), and Value (V). These projections are computed using stationary weight matrices  $W_Q$ ,  $W_K$ , and  $W_V$  of dimension  $[D, d_k]$ , where  $d_k$  is a (separate) weight dimension per attention head. This results in tensors  $Q = X \cdot W_Q$ ,  $K = X \cdot W_K$ , and  $V = X \cdot W_V$ , each with dimension of  $[B, N, d_k]$  per head. Given  $H = D/d_k$  heads, the dimension for each Q, K, and V becomes  $[B, H, N, d_k]$ .

**2. Logit Calculation:** The core of the attention mechanism is computing the logit scores  $L$  by performing a dot product between the query and key:  $L = Q \cdot K^T$ . This results in a score tensor of dimension  $[B, H, N, N]$ . The quadratic nature of this operation,

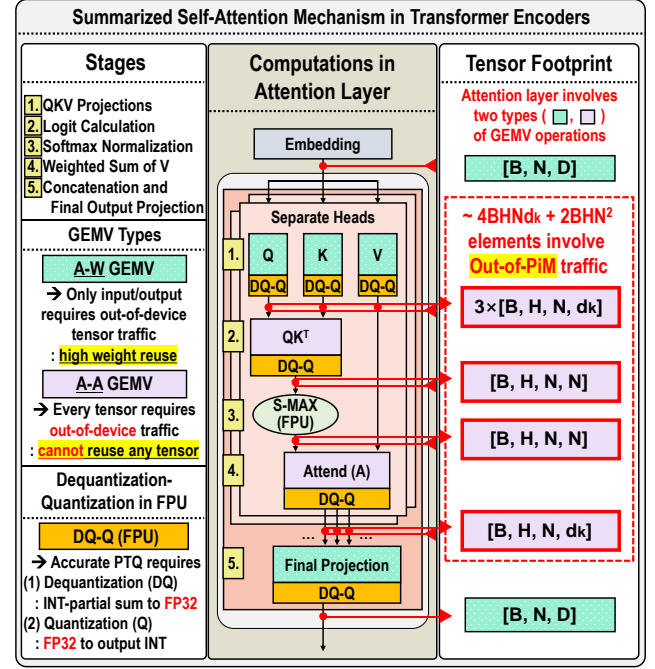


Figure 3: Overview of self-attention, with operation type and tensor traffic analysis. Self-attention comprises multiple linear projection layers with two types of GEMV operations, connected by Softmax, dimensional adjustments (transpose and concatenation), and DQ-Q operations. Unlike QAT, the DQ-Q process and nonlinear layers in PTQ create a deadlock between hardware overhead and tensor traffic.

requiring  $O(N^2d)$  operations per head, significantly contributing to the tensor traffic and computational load.

**3. Softmax Normalization:** The computed logits are scaled and passed through the Softmax layer to normalize the summed score. This normalization step, involving nonlinear operations, incorporates tensor traffic of  $[B, H, N, N]$ .

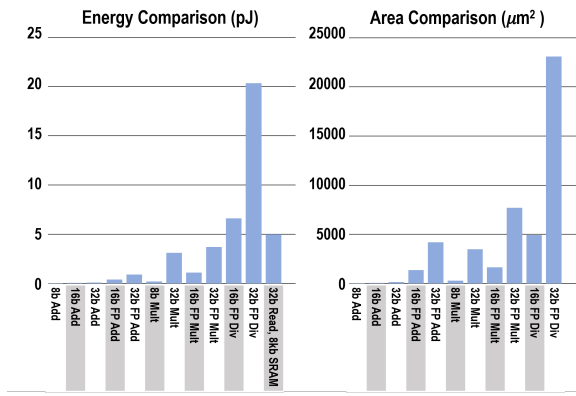
**4. Weighted Sum of Values:** The normalized attention scores are then used to compute a weighted sum of the value (V) vectors, yielding an output tensor of dimension  $[B, H, N, d_k]$ .

**5. Concatenation and Final Output Projection:** Finally, the outputs from all heads are concatenated and projected together once again, using a weight matrix ( $W_O$ ) to produce the final output tensor of  $[B, N, D]$ .

- **DQ-Q Process and FPU:** Not only the Softmax process involving nonlinear functions and accurate divisions, but also each of the stages (from step 1. to 5.) with PTQ necessitates the DQ-Q process, involving high-ENOB ADC, division, and FPUs. These requirements easily deteriorate the efficacy of PiM, motivating us to design a PiM-based accelerator that alleviates these limitations.

## 2.3 Kernel Fusion and Self-Attention Inference Optimization

The unique characteristics of the self-attention mechanism have delivered remarkable performance; however, they also introduce quadratic computational load and memory traffic, prompting the



**Figure 4: We alleviate not only the reliance on high-ENOB ADCs and FPUs but also the need for division arithmetic in our PiM hardware.**

development of various innovative techniques to alleviate these limitations. Alike PTQ, kernel (operation) fusion techniques also effectively achieve the given goal without impacting the model inference accuracy [2, 13, 28]. Kernel fusion techniques combine operations involving massive intermediate tensor traffic, reducing memory access overhead for these intermediate tensors.

The out-of-PiM tensor traffic for the self-attention computations without any on-device kernel fusion technique can be expressed as:

$$\begin{aligned} &\text{Original (Out-of-Device Tensor) traffic} \\ &= B \times N \times D \text{ (in)} + B \times N \times D \text{ (out)} \\ &+ 2 \times B \times H \times N^2 + 4 \times B \times H \times N \times d_k \text{ (in-and-out)}. \end{aligned}$$

As previously mentioned, PiM devices are only suited for processing BLAS layers on-device. Hence, without adequate optimization, the kernel fusion with their inherent limitations - high-ENOB ADCs and FPUs - deteriorate the effectiveness of PiM architectures.

Subsequently, our FLARE architecture proposes innovative quantization and nonlinear-layer processing techniques enabling the end-to-end kernel fusion with low hardware overhead, where the revised tensor traffic with our technique is:

Revised Tensor Traffic =  $2 \times B \times N \times D$  (in) +  $B \times N \times D$  (out), where the quadratic  $O(N^2)$  tensor traffic reduces to a linear  $O(N)$ .

### 3 Motivations

Our study reveals that the effectiveness of PiM architectures is notably hindered by the high-ENOB ADC and FPUs. Additionally, among various arithmetic operations required for the execution of self-attention, division arithmetic - not only in FP but also in integer formats - hinders the hardware efficiency, as illustrated in Fig. 4. Thus, our optimization strategy tackles these challenging overheads while preserving accuracy and performance, focusing on the following principles:

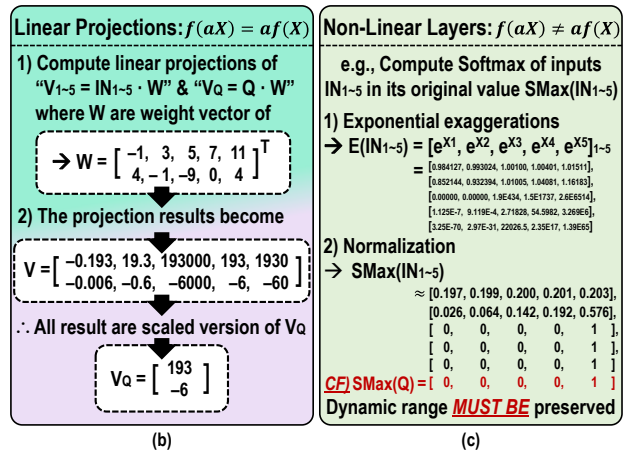
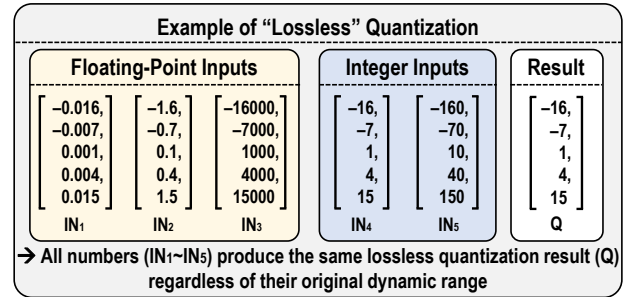
- **Accurate Quantization**
  - Lossless quantization: Our method must accurately represent quantized values even without FP or division arithmetic.
  - Outlier preservation: Attention layers often generate channels with extreme outliers, which standard integer quantization in

PTQ cannot capture well. To address this, we use token-wise quantization to reflect these outliers accurately.

- **Accurate Nonlinear (Non-BLAS)-Layer Executions**
  - Transformer models depend heavily on nonlinear layers, which require complex exponentiation and division operations. Our approach preserves the exponential trend and proportional relationships between values without FP or division arithmetic.
- **PVT-Robust and Error-Resilient AMS-PiM**
  - Despite its high efficiency, AMS-PiM has frequently been criticized for computation errors due to PVT variations. However, design solutions for those errors often conflict with the efficient exploitation of PiM devices. Our proposed techniques resolve this dilemma effectively.

In the following sections, we analyze these principles more deeply, before presenting our solutions as a breakthrough.

### 3.1 Motivation for PTQ and Non-BLAS Layer Optimizations



**Figure 5: (a) Even for a lossless quantization, values lose the exponent information. (b) The linear projections are consistent without exponent. (c) Softmax with the values shown at (a) - the absence of exponents deteriorates the computation result of nonlinear layers.**

While there are numerous quantization methods that we may not be able to cover in this literature, the fundamental steps for the accurate yet efficient quantization processes are as follows: (1) identify the minimum and maximum values among the given inputs ( $x_i \in X$ ), (2) for n-bit quantization, define a “quantization step (S),”



by dividing the minimum-maximum range by  $2^n$  levels, and (3) divide  $x_i$ 's by  $S$  to obtain the quantized values.

Along the process, as illustrated in Fig. 5-(a), we say a quantization is "lossless" when the ratios between quantized values retain the original proportions. That is:

$$\text{quantize}(x_i)/\text{quantize}(x_j) \approx x_i/x_j, \forall x_i, x_j \in \mathbf{X}. \quad (1)$$

However, the original exponent is not retained, even when "lossless" quantization is achieved using FPUs. Nonetheless, the issue of omitting exponents is less severe in linear projection operations (as depicted in Fig. 5-(b)), as operations keep their inherent quality without exponent information, i.e.,

$$a \times f(\mathbf{X}) = f(a \times \mathbf{X}). \quad (2)$$

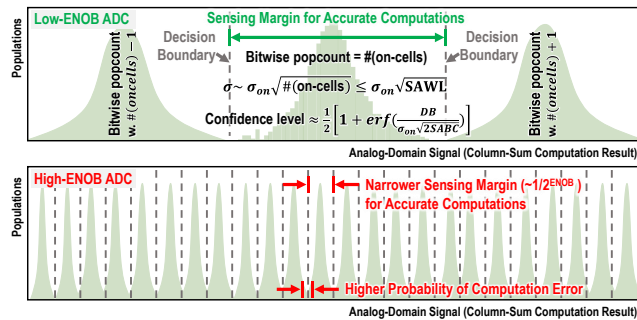
On the other hand, nonlinear layers are completely disrupted when the exponent information is omitted, as shown in Fig. 5-(c). That is,

$$a \times f(\mathbf{X}) \neq f(a \times \mathbf{X}). \quad (3)$$

Some recent studies simplify FP-based computations [9, 32, 37] to integer-based lightweight alternatives with satisfactory reliability. However, this approach can reduce the accuracy of PTQ-based inference, from the lack of retraining to adapt the model for these approximations.

In response, in section 4.2 and 4.3, we propose an integrated quantization-computation method that preserves the exponent for nonlinear layers while replacing division operations only with parsing and bit-shifting. While prior works [4, 20] introduced capturing the MSB's position, utilizing parsing(s), they rely on group-wise quantization, incurring large storage overhead and limited scalability. In contrast, our method combines token-wise quantization with VDR-Softmax in a fused manner, temporarily storing only a single-token value in compact registers and discarding them after processing, and linking quantization to penetrate through the end-to-end attributes of each stages in attention layer.

### 3.2 Motivation for Accurate and Efficient AMS Computation



**Figure 6: AMS-PiM with high-ENOB ADCs are more susceptible to computation errors due to PVT variation, resulting from reduced sensing margins.**

As mentioned in section 1, transformer models with PTQ demand a high precision for partial sum, easily exceeding  $\geq 18$  bits with  $d_{model} \geq 1k$  and with activation/weight operands of  $\geq 4$ bit. This

high-precision summation, for which direct quantization is infeasible, requires ADCs with higher ENOBs: this increases not only area/energy overhead but also the likelihood of errors.

Ensuring precise analog computation for PTQ requires the number of decision boundaries to exceed the distinct analog signal levels. This requirement is fundamental in PTQ, as accurate quantization relies on precise summation and digital conversion of the analog values. However, achieving this narrows the sensing margin of ADCs, which worsens error resilience, as depicted in Fig. 6. Errors occur when analog signals cross decision boundaries during analog-to-digital conversion, and this challenge intensifies as sensing margins (the distance between decision boundaries) narrow.

The computational error resilience, which prompts us to our proposed technique, can be analyzed as follows. Analog signals are proportional to the number of "on-cell"s generating on-current within the activated word lines (WLs), as illustrated in Fig. 2. Recent studies [3, 30, 56] show that reducing the activated "on-cell"s along the partial sum process reduces the error rates. Meanwhile, the number of Simultaneously Activated WLs (SAWL) determines the direct upper limit of the "on-cell"s contributing to the analog summation. Therefore, we can also limit the SAWL to limit the number of active "on-cell"s and segment unit computations with shorter input vectors for more reliable computation. This approach reduces the required ENOB of ADC, lowering the area/energy overhead of ADCs by  $2^{\text{ENOB}}$ , significantly relaxing the hardware complexity [12]. By integrating strategic segmentation reducing ADC overhead with error mitigation techniques described in section 4.4, FLARE can significantly improve PVT robustness.

### 3.3 Massive GEMVs with Quadratic Latency Bottleneck

As delineated in Section 3.2, the error-free analog computation can be achieved by limiting the maximum number of SAWL and segmenting computations over multiple cycles with sliced inputs (e.g., dividing  $D$  in Fig. 2 into  $D/N$  using  $N$  cycles). However, this sliced processing – with a constrained vector length – results in significantly more processing cycles. Furthermore, the increasing model size and the resultant quadratic GEMV demand in the attention layer further exacerbate the suboptimal performance.

For example, consider a length( $l$ )-restricted GEMM operation where  $\text{SAWL} \leq l=8$ , with:  $D = 1024$  hidden states and  $N = 512$  tokens per batch - resulting in 512 (=N) GEMVs -, using 8-bit integer representation (Bit-Precision,  $BP=8$ ) for both input and weight values. This necessitates  $D/l = 1024/8 = 128$  cycles to process a GEMV for each bit-serial vector of a token. It involves partitioning each bit-serial part of the input token into 8-WL chunks to limit  $\text{SAWL} \leq 8$ . Subsequently, the bitwise GEMV operations of  $D/l (=128)$ -cycles-per-bit are required for  $N \times BP = 512 \times 8 = 4,096$  times to process every GEMVs of bit-vectors and tokens composing the aforementioned GEMM, culminating in  $(D/l) \times N \times BP = 524,288$  cycles to process a GEMM of an embedding input with static weight matrices.

Assuming 10ns per bitwise GEMV, which is a fast figure for an AMS-PiM array, a GEMM operation for generating Q, K, or V would take approximately 0.5 milliseconds (ms). Other steps involving activation-activation GEMV with quadratic computational

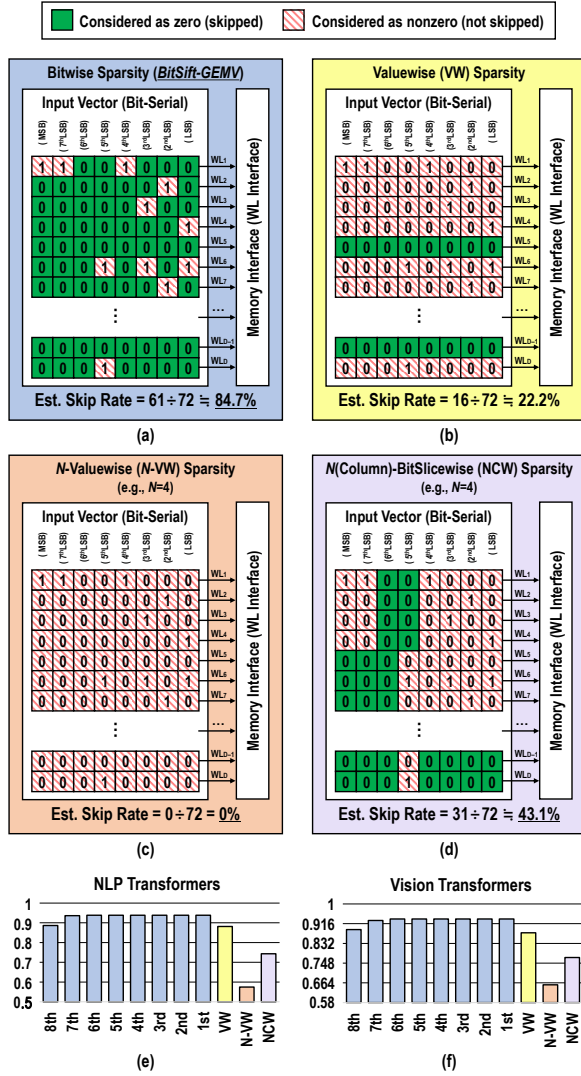


Figure 7: Comparison of sparsity measured in various granularity. (a)~(d) visualizes numerous sparsities where the same 8-bit-integer vectors are bit-serially inputted. Sparsities are measured and exploited; (a) bitwise manner, (b) valuewise (VW) manner (when 8-bit integer is of complete “0”s), N-valuewise (N-VW) manner, and (d) N-column-BitSlicewise manner (NCW, when a part of bitwise input is all “0”s in column direction). The activation sparsities are measured and averaged with input, Q, K, V, and output layers in transformer blocks for (e) NLP task and (f) vision task.

complexity and nonlinear layers may induce substantially greater latency, rendering the PiM device noncompetitive due to poor latency performance. Consequently, the latency for processing only the linear projections across the attention layers in tens of encoder blocks could be approximately in the order of 100 ms.

To address the latency issue, we take advantage of the high “bit-wise” sparsity found in the activation values of bitwise GEMVs in the attention layer. Our findings suggest collecting only “1”s along a bitwise embedding input can significantly boost GEMV - and

GEMMs composed of GEMVs - operations. Our investigation visualized in Fig. 7-(e), (f) shows that when split into bitwise elements (Fig. 7-(a)), the activation matrices exhibit much higher sparsity than when measured in coarser value grains (Fig. 7-(b) ~ (d)). With our findings, we propose BitSift-GEMV technique, where all bitwise “0”s of bit-serial fetched input activations are skipped from GEMV within AMS-PiM arrays.

In section 4.4, we propose a hardware-oriented technique that designates the longest combination(s) of parsed input slices, incorporating a fixed, desired number of “1”s. By parsing and merging the bit-serial-fetched token vectors, we can significantly stretch the length of a processible input vector for a unit GEMV. Furthermore, our proposed technique fully exploits the highest utilization of the designed ADCs, stabilizing its operation by limiting flexibility.

## 4 FLARE Architecture

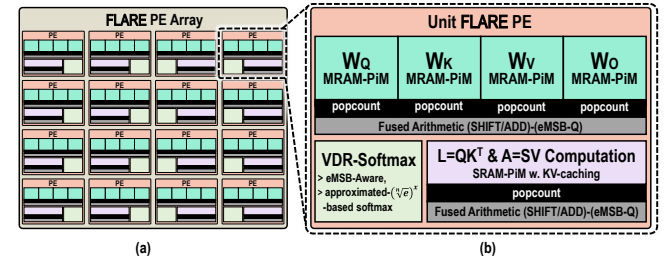


Figure 8: (a) Overall FLARE Architecture. (b) Hardware configurations of each FLARE PE.

Building upon our findings and discussions, we present our FLARE architecture — an FP- and division-less, end-to-end attention accelerator based on MRAM-SRAM hybrid AMS-PiM with low-ENOB ADCs. The architecture overview is visualized in Fig. 8. Our proposed architecture ensures accurate, reliable, efficient, and fast on-device attention-layer computations and is validated at the post-layout level using a 28nm FD-SOI process.

### 4.1 MRAM-SRAM Hybrid AMS-PiM Design

The basement of our system - MRAM-SRAM hybrid AMS-PiM design - utilizes the intrinsic advantages of each memory for two distinct types of GEMVs: activation-weight (A-W) and activation-activation (A-A) GEMVs, during the DNN’s main, and PiM’s core; BLAS accelerations. The MRAM-PiM are well-suited to A-W-pair GEMVs, owing to its non-volatile property which allows for significant weight reuse. Furthermore, MRAM maximizes array-level parallelism with a small memory cell size ( $\sim 1/3$  of SRAM). On the other hand, A-A-pair GEMVs, which require both operands to be dynamic, are handled using SRAM-PiM devices. SRAM offers the advantages of low write energy and high write speed compared to MRAM, making it ideal for A-A GEMVs.

The size and number of arrays are configured to fully encompass all parameters and computations of the self-attention layers, which is a fundamental requirement for achieving the end-to-end acceleration of the target model. The following paragraphs and Fig. 9-(a) outline the minimum array configuration requirements for implementing our FLARE architecture, while our specific design choices are summarized in TABLE 1.

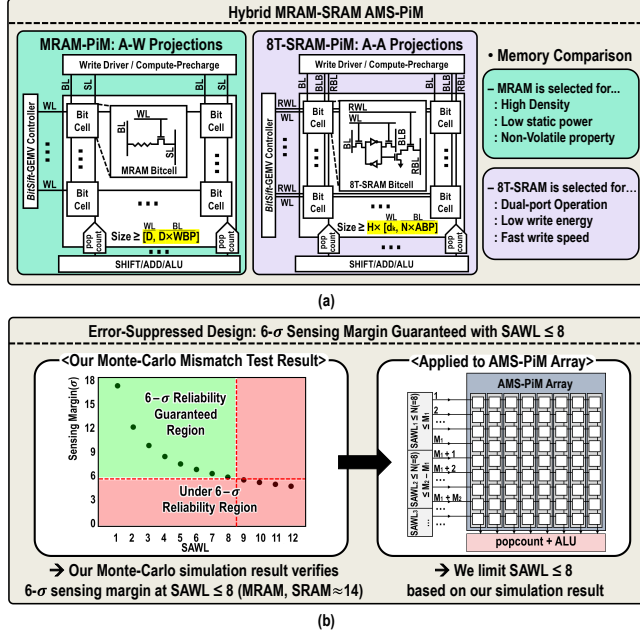


Figure 9: (a) Attribute and size requirements of each memory type, and (b) Monte-Carlo-simulation-guided SAWL selection for lossless computations.

First, the MRAM-CiM arrays for A-W GEMVs should store  $D^2$  weight elements for each query-, key-, value-, and the final-output projection layers. For each array, the total number of WLs must be larger than the hidden dimension  $D$  so that the input tokens can be full-parallelly fetched, and the total number of BLs should be larger than  $D \times WBP$  (weight-bit-precision) to independently store and compute the multi-bit weights at independent BLs.

Second, for the SRAM-PiM array, the number of WLs should be guaranteed so that it can handle  $D/H = d_k$  dimensions, while  $H$  independent arrays are needed to handle multi-head projections. The columns of the SRAM(s) must store  $N$  weight features, requiring  $N \times IBP$  (IBP: input-activation-bit-precision) BLs.

Meanwhile, to ensure a stable, error-free computation in AMS-PiM devices, we limit  $SAWL \leq 8$  to guarantee stable, 6-σ error-free computation, which is verified by Monte-Carlo simulation using our AMS-PiM arrays, where the results are summarized in Fig. 9-(b). This configuration effectively ensures lossless computation, however, for the latency risk tackled in section 3.3, we propose our fast, accurate, and efficient BitSift-GEMV method in section 4.4.

## 4.2 Dequantization-Free PTQ Technique

In this section, we propose a dequantization-free PTQ that penetrates all computations within the attention layer, replacing the FP and division arithmetics while retaining accuracy. As previously discussed, our proposed quantization method enables lossless integer quantization using only shifting and parsing, accommodating important attributes discussed in section 3.

As discussed in section 3.1, an integer quantization is “lossless” when the requirement from equation 1 is satisfied, which occurs if the quantization step size is smaller than that derived from floating-point representations. This implies that the original input can be

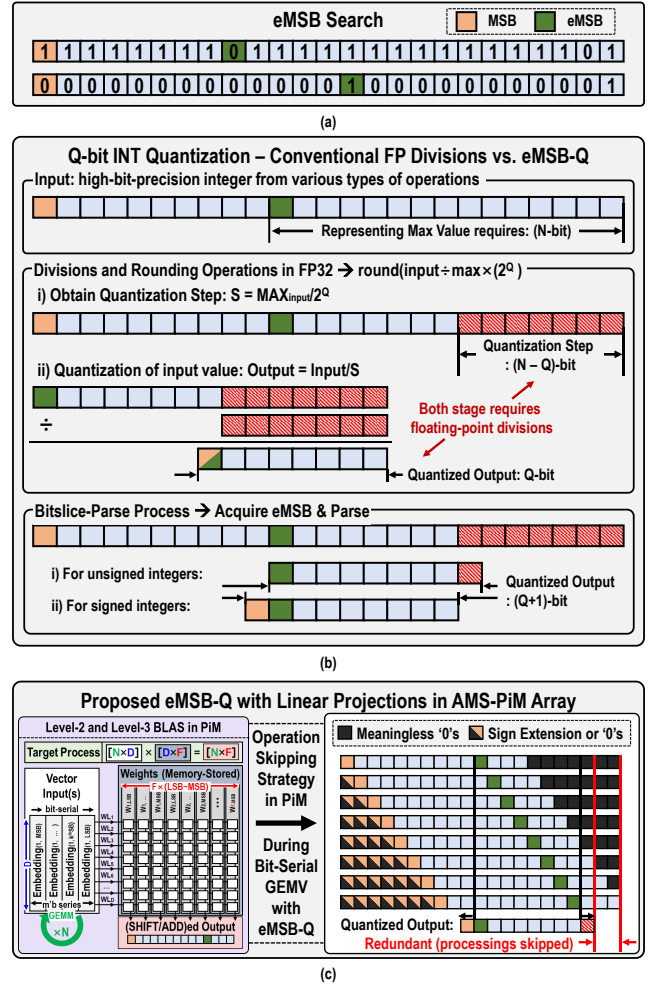


Figure 10: Our FP- and division-less integer quantization method, eMSB-Q. (a) Definition of eMSB for negative and positive (or unsigned) numbers. (b) eMSB-Q achieves lossless quantization at the expense of 1 more bit. (c) When applying our eMSB-Q for the AMS-PiM array, bit-serial inputs also benefit from skipping irrelevant computations.

divided by any  $2^n$ , as long as  $2^n < S$ , where  $S$  represents the quantization step size used in FP-based quantization. Therefore, we decided to replace the dequantization and division process, as depicted and compared with the conventional method, in Fig. 10-(a) and (b). Our method, termed effective-MSB-based Quantization (eMSB-Q), ensures numerical integrity while replacing the dequantization and quantization-step acquisition process with eMSB detection, and division operations with arithmetic shifting and parsing. The eMSB-search process simply identifies the location of the actual MSB within a group of bitwise values, eliminating the need for sorting to determine the maximum “value” and its “index.”

To prevent outliers from being clipped during eMSB-Q, we apply per-token quantization for  $Q, QK^T$ , and the final output. This approach leverages that each token vector in the  $Q$ -matrix is independently processed throughout the self-attention mechanism.

**Algorithm 1** VDR-Softmax

---

**Input:**  $x [Q_I-1:0][(N-1):0]$ ,  $n_e$   
 $N$  = number of the input elements  
 $Q_I$  = input bit precision,  $Q_O$  output bit precision  
 $n_e = \sum(\text{MSB-eMSB})$  over input  $\rightarrow Q \rightarrow QK^T$

**Parameters:**  
 $a, b, c, S, l \leftarrow a(n_e), b(n_e), c(n_e), S(n_e), l(n_e)$   
 $\Rightarrow$  once for  $Q_I$ , incorporates exponent information.  
 $\Rightarrow$  LUT-processed, 100 bytes

**Output:**  $x_{Softmax} [Q_O-1:0][(N-1):0]$

---

**function** VDR\_iPOLY( $r, n_e$ )  
 $x_{POLY} \leftarrow r \times (r + b) + c$   
 $S_{POLY} \leftarrow a \times S$   
**return**  $x_{POLY}, S_{POLY}$

**function** VDR\_iEXP( $x_{sub}, n_e$ )  
 $Q \leftarrow \text{clip}(\text{int}(X_{sub}/l), 2 \times Q_I)$   
 $r \leftarrow x_{sub} - Q - l$   
 $r_{POLY}, S_{POLY} \leftarrow \text{iPOLY}(r, S)$   
 $r_{EXP} = r_{POLY} \ll Q$   
 $S_{EXP} = S_{POLY} \gg Q$   
**return**  $r_{EXP}, S_{EXP}$

**function** VDR\_Norm( $x, S$ )  
 $x_{sub} = x - \max(x)$   
 $r_{EXP}, S_{EXP} = \text{iEXP}(x_{sub}, S)$   
 $x_{Softmax} = \text{eMSB-Q}(r_{EXP})$   
**return**  $x_{Softmax}, S_{EXP}$

---

However, KV generations - which must preserve global context - involve storing all vectors of  $N$  tokens. It differentiates the parsing strategy: we parse bits starting from MSBs rather than eMSBs, using a longer bit to reserve a wider dynamic range for the global context.

Note that, the inherent quality of linear operations is kept with eMSB-Q. However, to save the distinct attributes within the non-linear layer using our per-token quantization process, we transfer the eMSB information to the VDR-Softmax block, enabling it to incorporate exponent data during processing. This step is essential for the precise functioning of the Softmax layer, which relies on accurate exponent handling, unlike linear layers. In the following section, we further describe integer-processed Softmax for PTQ.

### 4.3 Proposed VDR-Softmax Fused with eMSB-Q

The simplified yet accurate implementation of the Softmax layer in PTQ requires a precise understanding of its characteristics and a more sophisticated integer-processing technique. The implementation involves several key steps, outlined in the Algorithm 1. Our approach utilizes per-token eMSB information to ensure accurate yet FP-less processing and uses only shifting and parsing of integers to normalize and quantize the score into integer format.

While accommodating exponent information in processing the nonlinear layer is critical, as visualized in Fig. 5-(c), directly multiplying/dividing the integer  $x$  makes the number exceed the representable range of the integers with designated bit-precision, leading to loss of information. The key to handling Softmax processing with integer arithmetic lies in adapting the exponent indirectly while approximating the  $e^{x/n}$  function, where  $x$  is an integer-format input

value and  $n$  is a representation of the exponent. To accomplish that, we adjust the base ( $e$ ) of the exponential function from  $e$  to  ${}^n\sqrt{e}$ , rather than adjusting the integer-represented  $x$  in order to avoid the loss of information. Using well-established,  $e^x$ -approximating functions from previous works [32, 37]. This adjustment is accomplished by modifying the internal parameters of the proposed method, without requiring division arithmetic.

Additionally, the base adjustment along the exponentiation process - from  $e$  to  ${}^n\sqrt{e}$  - is performed per token, using eMSB data from the eMSB-Q stage. Since Softmax is computed per token, aligned with our quantization approach (in section 4.2) and dataflow control mechanism (in section 4.5), we do not need to store eMSB positions for all Q vectors, enhancing our architecture’s scalability.

After the exponentiations, we apply eMSB-Q to a group of exponentiated outputs for each token, maximizing the utilization of the dynamic range provided by a  $Q_O$ -bit (Softmax output precision) integer. This approach avoids division while preserving the original proportions of the values, ensuring that the maximum value is mapped to the highest possible representation, with sufficient range allocated for smaller values.

Consequently, our VDR-Softmax achieves both efficiency and accuracy, making it well-suited for end-to-end, integer-based acceleration within our PTQ-based attention accelerator architecture. This approach not only retains the critical properties of the exponentiation and divisions in Softmax, but also alleviates the computational overhead, leading to significantly enhanced efficiency with secured performance.

### 4.4 BitSift-GEMV Processor

Along the end-to-end, on-device fusion of the self-attention layer, our BitSift-GEMV technique for AMS-PiM significantly boosts sparse GEMV, which composes most of the computations in the attention layer. BitSift-GEMV is motivated by two key factors: (1) the highlighted bitwise sparsity of activation data illustrated in Fig. 7 and (2) the constant-maximum number of “1”s (=8) WL activations, which allows the highest-utilization-rate operation and strengthened robustness of ADC designs.

Existing AMS-PiM designs require flexibility in the number of processible SAWL as input vectors have a random number of “1”s along the bit-slice. One thing to note here is that a flexible SAWL can lead to worse computation reliability with higher overhead for the ADC design (Fig. 11-(b)). For SAWL-flexible configurations, the ADCs should handle a wider, variable dynamic range, higher resolution, and varying threshold during the analog-to-digital conversions. This also compromises the utilization rate when the actual SAWL lowers, wasting power/area implemented for higher SAWL. Therefore, our design sets “SAWL=MAX” for all time without flexibility, by incorporating dummy-“1” inputs to maintain a fixed, maximum number of “1”s.

To maintain a constant number of SAWL at 8 (= maximum SAWL that allows 6- $\sigma$  reliability), the SAWL<sub>D</sub> controller in Fig. 11-(a) supplements the SAWL with additional “1”s, when the input vector has fewer “1”s than 8. Specifically, the SAWL<sub>D</sub> controller activates SAWL<sub>D</sub> = (8-SAWL) WLs at the bottom of the array. This is achieved by incorporating a dummy array and WL interfaces consisting of seven rows at the bottom. The memory cells within



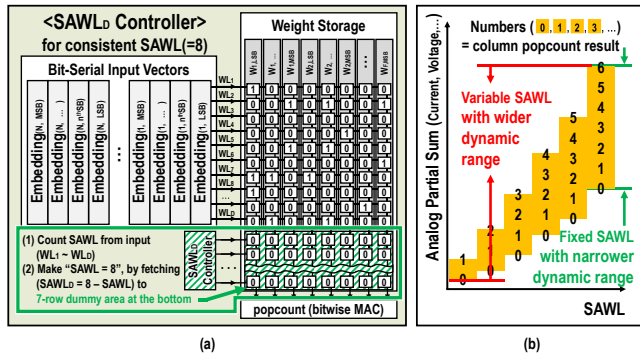


Figure 11: (a) SAWL<sub>D</sub> controller enabling a fixed number of SAWL(=8) when the input data is insufficient of “1”s. (b) Relation between SAWL and the dynamic range variation of analog-signal-domain popcount. Limiting flexibility with SAWL relaxes the ADC-design challenges for AMS-PiM.

these dummy rows are all set to “0” (off-cells), emulating computations with dummy inputs. The array configuration serves two purposes: (1) isolating dummy operations - it prevents the dummy array from affecting the actual computation output, and (2) enhancing variation resiliency - by consistently returning “0”s, the dummy array area improves the system’s robustness against variations. I.e., While processing only “1”s along the input vector, the bitwise “0”s along the vector(s) are skipped from the actual computation.

The fixed-SAWL processing incorporating SAWL<sub>D</sub> controller employs a two-step approach. First, the input vector is scanned to identify the longest slice containing eight “1”s. The slice is then selected for processing. Second, for the tail end of a vector with fewer “1”s (or, for a very sparse input vector), the SAWL<sub>D</sub> processor supplements the existing “1”s with additional ones from the dummy array. This ensures a consistent maximum SAWL for all computations, regardless of the input vector’s density. By maintaining a constant SAWL of 8, this method fixes the input dynamic range for ADC and enables lower area/energy/error overhead, even with varying input sparsity.

To support the SAWL<sub>D</sub> controller by identifying the longest segment(s) of the input with the largest number of SAWL that remain constrained of being  $\leq 8$ , we introduce our BitSift-GEMV controller, depicted in Fig. 12. This strategy introduces a novel approach for highly efficient sparse GEMV processing, achieving low latency and minimal area/energy overhead. Instead of a hypothetical brute-force method that might scan the entire input vector bit by bit - resulting in significant latency penalties as vector length and sparsity increase - the BitSift-GEMV controller leverages a hierarchical design with high parallelism. This includes (1) a local-pop controller (LPC) and (2) a global-pop controller (GPC), where “pop-count” refers to the number of “1”s in the input vector.

(1) The LPC efficiently counts the number of “1”s within short, fixed-length slices of the input vector using dedicated “pop detector” and “popcount” blocks. This localized processing ensures low-latency computation of the popcount within each slice. (2) The GPC aggregates the popcounts from LPCs to rapidly identify the highest possible number of “1”s, constrained to a maximum of 8, for further processing. Once aggregation is complete, the GPC

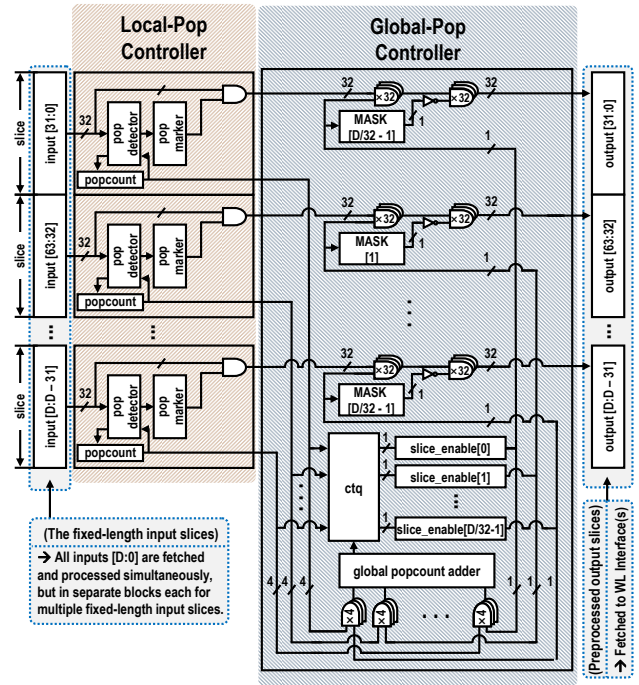


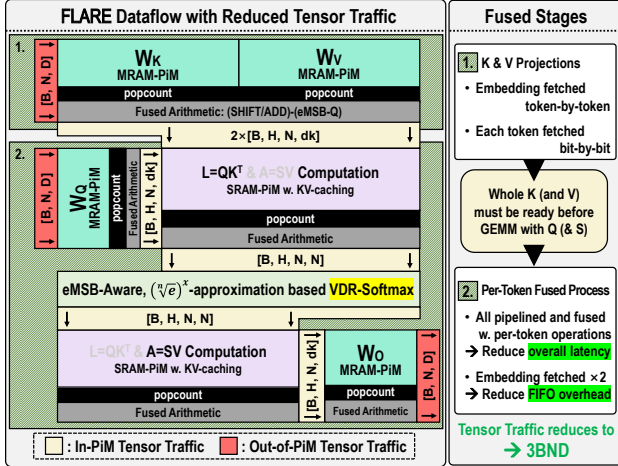
Figure 12: Our BitSift-GEMV controller parses, selects, and fetches the longest portion(s) of the input - that contains designated SAWL - to WL interface(s).

activates the “MASK” to fetch the corresponding vector to the output and W<sub>L</sub>s. If the total number of “1”s identified by the GPC is fewer than 8, the SAWL<sub>D</sub> controller (Fig. 11-(a)) supplements the input by adding “1”s from the dummy array before forwarding it to the W<sub>L</sub>s of PiM array. This hierarchical and parallel architecture enables our AMS-PiM arrays to sustain high performance, even as input vector lengths and sparsity increase. The design is specifically optimized for extremely sparse inputs, especially whose popcount within each LPC’s slice remains  $\leq 8$ . This emphasis on sparsity aligns with the activation values frequently observed in neural networks, as illustrated in Fig. 7-(e), (f).

However, recognizing that real-world data can exhibit varying sparsity patterns, the BitSift-Controller also incorporates dedicated circuitry to handle denser input segments. That is, our BitSift-GEMV controller dynamically adjusts the processing flow when an LPC encounters eight or more “1”s within its slice, or when the GPC detects a global popcount exceeding 8. This dynamic adjustment mechanism is as follows. When the “pop detector” block in LPC detects  $8 \times “1”$ s before reaching the end of the sliced input, the inputs are marked from the beginning to the point where the 8-th “1” is found, using the “pop marker” block. The marked section is then fetched first, consuming a column-sum cycle in the AMS-PiM array. After that, the marker marks from the next section of the inputs to be processed - excluding the section fetched so far - and continues until the process reaches the end of the LPC’s processible input section. After the LPC-level sparsity requirement (popcount < 8) is all met, in the GPC, when the global popcount adder returns  $> 8$ , the ctq (compute-token queue) circuit block controls the “MASK”s, gating the LPCs from fetching the input to the PiM array so that

the LPCs of having summed popcount under 8 can be only fetched. Thus along the input vector of length  $[D]$ , it takes processing cycles up to  $\text{roundup}(\frac{D \times (1 - \text{bitwise\_sparsity})}{8})$ . Therefore, our BitSift-GEMV technique reduces the processing latency by  $(1 - \text{bitwise\_sparsity})$ , fully utilizing the finest grain of sparsity.

#### 4.5 Reduced Tensor Traffic



**Figure 13: Visualized Dataflow and tensor traffic of FLARE: QKV generation, on-the-fly processing, and fused operations.**

Conclusively, our FLARE architecture minimizes tensor traffic in attention mechanisms by fusing the end-to-end attention layer with combined novel techniques. The dataflow of our FLARE architecture (depicted in Fig. 13) can be understood as follows: (1) K & V Projections and (2) end-to-end fused, per-token ( $Q$ - $L$ - $A$ - $O$ ) process.

(1) K & V projections: This stage involves A-W GEMM (a series of GEMV) operations on input tokens with weight matrices  $W_K$  &  $W_V$ , requiring input streaming to generate the complete KV matrices.

(2) Per-token fused process: After generating KV matrices, the end-to-end-fused attention process is handled, requiring a second input stream-in. Rather than storing an input in an internal buffer, we stream data twice for improved area efficiency and scalability.

Hence, the revised tensor traffic can be represented as:

$$\text{Revised Tensor Traffic} = 2 \times B \times N \times D \text{ (in)} + B \times N \times D \text{ (out)},$$

achieving substantial tensor traffic reduction by leveraging our proposed techniques.

### 5 FLARE Evaluations

Our proposed FLARE architecture and its design components are all embedded into a compact ASIC with superior energy and latency performance per token. We validate our design through meticulous circuit simulations, including Monte Carlo simulations (result summarized in Fig. 9-(b) and post-layout circuit simulations for physical-level feasibility, using our custom-designed ASIC on 28nm FD-SOI technology, and our design is summarized in TABLE 1.

We tested our FLARE using 8-bit integer-quantized models across different NLP (with GLUE benchmarks [52]) and vision (ImageNet [14] Classification) tasks, as well as with various transformer models [16, 17, 38, 49]. We compared the results with state-of-the-art

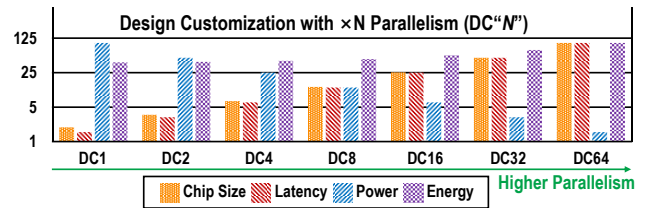
**Table 1: Summary of our FLARE design in 28nm process**

| FLARE Properties in 28nm Process                           |                         |                                   |  |
|--|-------------------------|-----------------------------------|--|
| Component  | Area (mm <sup>2</sup> ) | Power (mW)                        | Parameters & Specifications  |
| <b>A-W Linear Projection Modules</b>                       |                         |                                   |  |
| BitSift-GEMV Controller                                    | 0.003468                | 8.11                              | Total Bit Cells: 72×1024×1024  |
| MRAM + ADC   | 19.546                  | 14.4<br>(power gated unless used) |  |
| SHIFT/ADD  | 0.221                   | 2.47                              |  |
| eMSB-Q   | 0.0202                  | 0.213                             |  |
| <b>A-A Linear Projection Modules</b>                       |                         |                                   |  |
| BitSift-GEMV Controller                                    | 0.001156                | 3.6                               | Total Bit Cells: 512×64×1024   |
| SRAM + ADC   | 5.384                   | 45.2<br>(power gated unless used) |  |
| SHIFT/ADD  | 0.553                   | 6.24                              |  |
| eMSB-Q   | 0.11                    | 1.17                              |  |
| <b>VDR-SoftMax Module</b>                                  |                         |                                   |  |
| iPoly  | 1.17                    | 48.4                              | iPoly: 2 <sup>nd</sup> -order polynomial<br>iEXP with Max. integer bit inside: 24 <sup>b</sup> |
| iEXP   | 1.29                    | 18.6                              |  |
| VDR-Norm (eMSB-Q)  | 0.23                    | 2.29                              |  |
| <b>FLARE Architecture (PE with All Modules / PE Array)</b> |                         |                                   |  |
| FLARE  | 29.665                  | 150.8                             | Per Single Attention Layer<br>(*Approximated in 7nm process)                                   |
| Unit PE  | ( $\approx 1.86^*$ )    |                                   |  |
| FLARE  | 492.44                  | 2,535                             | 16 PEs for multi-layer processing<br>(*Approximated in 7nm process)                            |
| PE Array   | ( $\approx 30.78^*$ )   |                                   |  |

GPUs (Nvidia RTX 3090, RTX 4090, T4, and A100), as well as a PiM baseline where we excluded our proposed techniques.

Our experimental results validate the FLARE architecture’s diverse efficacy. The design exploration highlights the significance of customizing inner-array configurations and parallelism levels to optimize performance. The eMSB-Q and VDR-Softmax techniques maintain high inference accuracy while eliminating FP operations, thereby enhancing computational efficiency. Additionally, the BitSift-GEMV technique leverages bitwise sparsity to significantly accelerate GEMV operations. Collectively, these innovations establish our architecture as a robust and scalable solution for transformer acceleration, delivering substantial reductions in both latency and energy consumption.

#### 5.1 Hardware Customization Exploration



**Figure 14: Design study on array-level device parallelism. Larger values imply better characteristics.**

We conducted a design exploration study to understand how different array-parallelism configurations affect the overall performance of our proposed architecture. Fig. 14 shows the normalized metrics of various design choices, including chip size, latency, power, and energy consumption. For the summary in Table 1, our choice for the parallelism is 8× array (i.e., DC8), to show the design result with a medium benchmark.

### 5.2 Impact of eMSB-Q and VDR-Softmax

| Accuracy Results on Various Model & Tasks Benchmarks                                 |  |              |                                   |              |              |              |              |       |
|--|--|--------------|-----------------------------------|--------------|--------------|--------------|--------------|-------|
| ImageNet Classifications   |  |              | Natural Language Processing Tasks |              |              |              |              |       |
|  |  |              | Model                             | BERT-Base    |              | BERT-Large   |              |       |
| Model  | Method   | Accuracy (%) | Benchmark                         | Method       | Accuracy (%) | Method       | Accuracy (%) |       |
| ViT-S  | fp32   | 81.14        | MNLI-m                            | fp32         | 89.56        | fp32         | 91.80        |       |
|  | Conventional                                     | 80.69        |                                   | Conventional | 86.63        | Conventional | 89.50        |       |
|  | Proposed   | 81.08        |                                   | Proposed     | 89.70        | Proposed     | 91.72        |       |
| ViT-L  | fp32   | 84.28        | MNLI-mm                           | fp32         | 89.25        | fp32         | 91.70        |       |
|  | Conventional                                     | 83.10        |                                   | Conventional | 86.53        | Conventional | 89.40        |       |
|  | Proposed   | 84.23        |                                   | Proposed     | 88.98        | Proposed     | 91.51        |       |
| DeiT-T   | fp32   | 71.99        | QQP                               | fp32         | 92.21        | fp32         | 94.66        |       |
|  | Conventional                                     | 71.66        |                                   | Conventional | 90.30        | Conventional | 92.07        |       |
|  | Proposed   | 71.94        |                                   | Proposed     | 91.54        | Proposed     | 93.62        |       |
| DeiT-S   | fp32   | 79.61        | QNLI                              | fp32         | 94.66        | fp32         | 95.98        |       |
|  | Conventional                                     | 79.39        |                                   | Conventional | 91.87        | Conventional | 93.56        |       |
|  | Proposed   | 79.58        |                                   | Proposed     | 93.54        | Proposed     | 95.74        |       |
| DeiT-B   | fp32   | 81.60        | SST-2                             | fp32         | 96.49        | fp32         | 98.23        |       |
|  | Conventional                                     | 81.14        |                                   | Conventional | 94.25        | Conventional | 95.44        |       |
|  | Proposed   | 81.54        |                                   | Proposed     | 96.27        | Proposed     | 97.95        |       |
| Quantization Methods   |  |              | CoLA                              | fp32         | 62.42        | fp32         | 69.36        |       |
| • Conventional = Quantized INT8 with fp32 dequantization-quantization + fp32 softmax |  |              |                                   | Conventional | 61.88        | Conventional | 68.31        |       |
|  |  |              |                                   | Proposed     | 62.45        | Proposed     | 69.55        |       |
|  | • Proposed = INT(8+1) using eMSB-Q + VDR-Softmax |              |                                   | fp32         | 92.92        | fp32         | 94.04        |       |
| STS-B  |  |              |                                   | Conventional | 89.89        | Conventional | 91.28        |       |
|  |  |              |                                   | Proposed     | 91.71        | Proposed     | 94.06        |       |
| MRPC   |  |              | fp32                              | 92.72        | fp32         | 93.64        |              |       |
|  |  |              | Conventional                      | 90.19        | Conventional | 92.07        |              |       |
|  |  |              |                                   | Proposed     | 92.84        | Proposed     | 93.01        |       |
|  |  | RTE          |                                   |              | fp32         | 79.56        | fp32         | 88.03 |
|  |  |              |                                   |              | Conventional | 78.61        | Conventional | 86.13 |
|  |  |              |                                   |              | Proposed     | 79.61        | Proposed     | 88.00 |

Figure 15: Comparison of inference accuracy results: various NLP tasks and ImageNet classification tasks with different transformer models.

Fig. 15 presents the inference accuracy benchmarks across various tasks and models. The comparison includes the FP32 baseline, a traditionally-integer-quantized model utilizing the FP32-based DQ-Q process, and our proposed method employing eMSB-Q quantization combined with the VDR-Softmax technique. The results demonstrate that our approach sustains high accuracy while eliminating FP operations, thereby achieving substantial improvements in computational efficiency.

### 5.3 Impact of BitSift-GEMV

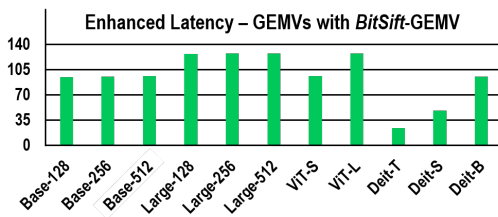


Figure 16: Average amount of how much GEMV operations were boosted. The boosting factors were almost directly the same as our anticipated values, revealing our BitSift-GEMV’s efficacy.

Fig. 16 presents the impact of our proposed BitSift-GEMV technique on the average number of boosting factors in GEMV operation cycles, compared to PiM baseline with fixed-length processed GEMV. Note that the boosting is marked without the effect of array

parallelism, and the measurement result is solely boosted by the BitSift-GEMV only. The boosting factor observed correlates closely with our anticipated values, demonstrating the effectiveness of exploiting bitwise sparsity to enhance GEMV processing speed.

### 5.4 Impact on Latency Performance

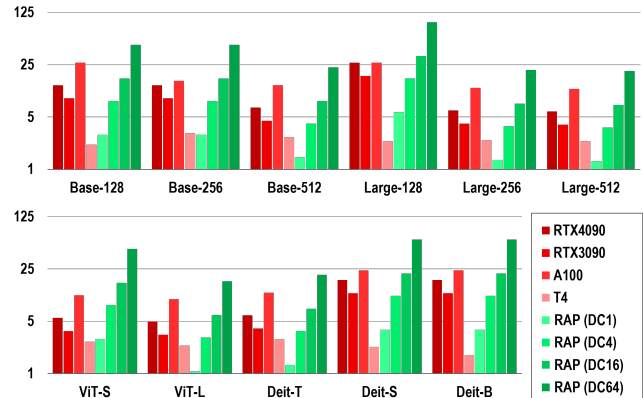


Figure 17: Comparison of normalized token/sec performance. The CiM Baseline was omitted due to its poor performance from significant latency with out-of-PiM tensor traffic and segmented GEMV.

We measured and compared the token/sec performance with various tasks, models, and hardware. Fig. 17 shows that our design achieves competitive latency performance, outperforming not only the PiM baseline but also famous SOTA GPUs. This improvement is primarily attributed to the integration of the end-to-end fusion technique and the BitSift-GEMV approach.

### 5.5 Impact on Energy Performance

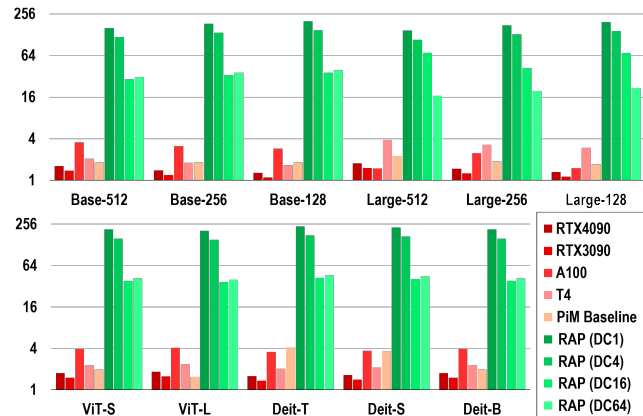


Figure 18: Comparison of normalized token/Joule performance.

Fig. 18 illustrates the energy efficiency of our FLARE architecture compared to various SOTA GPUs and a PiM baseline across diverse NLP and vision tasks. The results highlight that our architecture achieves outstanding energy efficiency, positioning it as a practical solution for energy-constrained applications. Notably, without our

on-device end-to-end kernel fusion technique, the reliance on FPUs or the substantial out-of-PiM tensor traffic significantly undermines the baseline PiM devices' inherent efficiency.

## 6 Discussion - Related Works

FLARE's design offers an accurate, fast, and efficient foundation for accelerating self-attention layers in encoder models. Beyond its standalone capabilities, FLARE seamlessly integrates with optimization techniques to further enhance performance without hardware revisions. For instance, FLARE complements FlashAttention [13] by processing tiles independently in a FLARE PE, effectively scaling to long sequences or oversized models while preserving its core strengths. Also, FLARE aligns with Dynamic Attention Modulation techniques [40], dynamically allocating resources based on input complexity to reduce redundant computations and optimize energy efficiency. Moreover, FLARE supports hot-expert routing [33] in Mixture-of-Experts models, efficiently handling intensive computations while reducing tensor traffic. Finally, representation compression techniques [25, 27, 34, 42] align with FLARE's processing to lower computational demands without sacrificing accuracy.

## 7 Conclusion

This work presents FLARE, an AMS-PiM-based architecture designed to overcome the computational and hardware bottlenecks of transformer models. By introducing dequantization-free PTQ, integer-only nonlinear processing, and BitSift-GEMV for sparse GEMV acceleration, FLARE achieves robust energy efficiency, error resilience, and computational performance. FLARE's hybrid MRAM-SRAM design enables on-chip processing of self-attention layers while eliminating the need for high-ENOB ADCs and FPUs, reducing area and power overhead. Experimental results confirm FLARE's superiority over GPUs and PiM baselines in latency, energy efficiency, and scalability, making it a practical solution for deploying transformers in diverse environments.

## References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altmenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).
- [2] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. 2016. Fused-layer CNN accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.
- [3] Tanner Andrusis, Joel S Emer, and Vivienne Sze. 2023. RAELLA: Reforming the arithmetic for efficient, low-resolution, and low-loss analog PiM: No retraining required!. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–16.
- [4] Anonymous. 2024. QRazor: Reliable and Effortless 4-bit LLM Quantization by Significant Data Razoring. In *Submitted to The Thirteenth International Conference on Learning Representations*. <https://openreview.net/forum?id=lwcnZmyojm> under review.
- [5] Ron Banner, Yury Nahshan, and Daniel Soudry. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. *Advances in Neural Information Processing Systems* 32 (2019).
- [6] Avishek Biswas and Anantha P Chandrakasan. 2018. CONV-SRAM: An energy-efficient SRAM with in-memory dot-product computation for low-power convolutional neural networks. *IEEE Journal of Solid-State Circuits* 54, 1 (2018), 217–230.
- [7] L Susan Blackford, Antoine Petit, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. 2002. An updated set of basic linear algebra subprograms (BLAS). *ACM Trans. Math. Software* 28, 2 (2002), 135–151.
- [8] Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).
- [9] Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Alberto Nannarelli, Marco Re, and Sergio Spanò. 2021. A pseudo-softmax function for hardware-based high speed image classification. *Scientific reports* 11, 1 (2021), 15307.
- [10] Wei-Hao Chen, Kai-Xiang Li, Wei-Yu Lin, Kuo-Hsiang Hsu, Pin-Yi Li, Cheng-Han Yang, Cheng-Xin Xue, En-Yu Yang, Yen-Kai Chen, Yun-Sheng Chang, et al. 2018. A 65nm 1Mb nonvolatile computing-in-memory ReRAM macro with sub-16ns multiply-and-accumulate for binary DNN AI edge processors. In *2018 IEEE International Solid-State Circuits Conference-ISSCC*. IEEE, 494–496.
- [11] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. 2016. 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)*. IEEE, 27–39.
- [12] Loai Danial, Nicolás Wainstein, Shraga Kraus, and Shahar Kvatinsky. 2018. Breaking through the speed-power-accuracy tradeoff in ADCs using a memristive neuromorphic architecture. *IEEE Transactions on Emerging Topics in Computational Intelligence* 2, 5 (2018), 396–409.
- [13] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems* 35 (2022), 16344–16359.
- [14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.
- [15] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems* 36 (2024).
- [16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).
- [17] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xi-aohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
- [18] Xinjie Guo, F Merrikh Bayat, M Bavandpour, M Klachko, MR Mahmoodi, M Prezioso, KK Likharev, and DB Strukov. 2017. Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology. In *2017 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 6–5.
- [19] Saransh Gupta, Mohsen Imani, Harveen Kaur, and Tajana Simunic Rosing. 2019. Nnpim: A processing in-memory architecture for neural network acceleration. *IEEE Trans. Comput.* 68, 9 (2019), 1325–1337.
- [20] Nguyen-Dong Ho and Ik-Joon Chang. 2023. O-2A: Ourlier-Aware Compression for 8-bit Post-Training Quantization Model. *IEEE Access* (2023).
- [21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2018. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research* 18, 187 (2018), 1–30.
- [22] Mohsen Imani, Saransh Gupta, Yeseong Kim, Minxuan Zhou, and Tajana Rosing. 2019. Digitalpim: Digital-based processing in-memory for big data acceleration. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*. 429–434.
- [23] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.
- [24] Zhewei Jiang, Shihui Yin, Jae-Sun Seo, and Mingoo Seok. 2020. C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism. *IEEE Journal of Solid-State Circuits* 55, 7 (2020), 1888–1897.
- [25] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).
- [26] Seungchul Jung, Hyungwoo Lee, Sungmeon Myung, Hyunsoo Kim, Seung Keun Yoon, Soon-Wan Kwon, Yongmin Ju, Minje Kim, Wooseok Yi, Shinhee Han, et al. 2022. A crossbar array of magnetoresistive memory devices for in-memory computing. *Nature* 601, 7892 (2022), 211–216.
- [27] Herve Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. <https://doi.org/10.1109/TPAMI.2010.57>
- [28] Sheng-Chun Kao, Suvinay Subramanian, Gaurav Agrawal, Amir Yazdanbakhsh, and Tushar Krishna. 2023. Flat: An optimized dataflow for mitigating attention bottlenecks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 295–310.
- [29] Arman Kazemi, Mohammad Mehdi Sharifi, Zhuowen Zou, Michael Niemier, X Sharon Hu, and Mohsen Imani. 2021. Mimhd: Accurate and efficient hyperdimensional inference using multi-bit in-memory computing. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE,



- 1–6.
- [30] Hyungjun Kim, Hyunmyung Oh, and Jae-Joon Kim. 2020. Energy-efficient XNOR-free in-memory BNN accelerator with input distribution regularization. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.
- [31] Hyeonuk Kim, Jaehyeong Sim, Yeongjae Choi, and Lee-Sup Kim. 2019. Nand-net: Minimizing computational complexity of in-memory processing for binary neural networks. In *2019 IEEE international symposium on high performance computer architecture (HPCA)*. IEEE, 661–673.
- [32] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*. PMLR, 5506–5518.
- [33] Taehyun Kim, Kwansoek Choi, Youngmook Cho, Jaehoon Cho, Hyuk-Jae Lee, and Jaewoong Sim. 2024. MoNDE: Mixture of Near-Data Experts for Large-Scale Sparse Models. *arXiv preprint arXiv:2405.18832* (2024).
- [34] Z Lan. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).
- [35] Hunjun Lee, Minseop Kim, Dongmoon Min, Joonsung Kim, Jongwon Back, Honam Yoo, Jong-Ho Lee, and Jangwoo Kim. 2022. 3D-FPIM: An extreme energy-efficient DNN acceleration system using 3D NAND flash-based in-situ PIM unit. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1359–1376.
- [36] Huize Li, Zhaoying Li, Zhenyu Bai, and Tulika Mitra. 2024. ASADI: Accelerating Sparse Attention Using Diagonal-based In-Situ Computing. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 774–787.
- [37] Zhikai Li and Qingyi Gu. 2023. I-ivit: Integer-only quantization for efficient vision transformer inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 17065–17075.
- [38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).
- [39] Yun Long, Taesik Na, and Saibal Mukhopadhyay. 2018. ReRAM-Based Processing-in-Memory Architecture for Recurrent Neural Network Acceleration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26, 12 (2018), 2781–2794. <https://doi.org/10.1109/TVLSI.2018.2819190>
- [40] Abhishek Moitra, Abhiroop Bhattacharjee, and Priyadarshini Panda. 2024. PIVOT-Input-aware Path Selection for Energy-efficient ViT Inference. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.
- [41] Sourjya Roy, Mustafa Ali, and Anand Raghunathan. 2021. PIM-DRAM: Accelerating machine learning workloads using processing in commodity DRAM. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 4 (2021), 701–710.
- [42] V Sanh. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
- [43] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. 2016. 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)*. IEEE, 14–26.
- [44] Xin Si, Jia-Jing Chen, Yung-Ning Tu, Wei-Hsing Huang, Jing-Hong Wang, Yen-Cheng Chiu, Wei-Chen Wei, Ssu-Yen Wu, Xiaoyu Sun, Rui Liu, et al. 2019. 24.5 A twin-8T SRAM computation-in-memory macro for multiple-bit CNN-based machine learning. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 396–398.
- [45] Xin Si, Jia-Jing Chen, Yung-Ning Tu, Wei-Hsing Huang, Jing-Hong Wang, Yen-Cheng Chiu, Wei-Chen Wei, Ssu-Yen Wu, Xiaoyu Sun, Rui Liu, et al. 2019. A twin-8T SRAM computation-in-memory unit-macro for multibit CNN-based AI edge processors. *IEEE Journal of Solid-State Circuits* 55, 1 (2019), 189–202.
- [46] Shrihari Sridharan, Jacob R Stevens, Kaushik Roy, and Anand Raghunathan. 2023. X-former: In-memory acceleration of transformers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, 8 (2023), 1223–1233.
- [47] Fang Su, Wei-Hao Chen, Lixue Xia, Chieh-Pu Lo, Tianqi Tang, Zhibo Wang, Kuo-Hsiang Hsu, Ming Cheng, Jun-Yi Li, Yuan Xie, et al. 2017. A 462GOPS/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory. In *2017 Symposium on VLSI Technology*. IEEE, T260–T261.
- [48] Baohua Sun, Daniel Liu, Leo Yu, Jay Li, Helen Liu, Wenhan Zhang, and Terry Torng. 2018. MRAM co-designed processing-in-memory CNN accelerator for mobile and IoT applications. *arXiv preprint arXiv:1811.12179* (2018).
- [49] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. 2021. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*. PMLR, 10347–10357.
- [50] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).
- [51] Naveen Verma, Hongyang Jia, Hossein Valavi, Yinqi Tang, Murat Ozatay, Lung-Yen Chen, Bonan Zhang, and Peter Deaville. 2019. In-memory computing: Advances and prospects. *IEEE Solid-State Circuits Magazine* 11, 3 (2019), 43–55.
- [52] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. 353–355.
- [53] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*. PMLR, 38087–38099.
- [54] Cheng-Xin Xue, Wei-Hao Chen, Je-Syu Liu, Jia-Fang Li, Wei-Yu Lin, Wei-En Lin, Jing-Hong Wang, Wei-Chen Wei, Ting-Wei Chang, Tung-Cheng Chang, et al. 2019. 24.1 A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 388–390.
- [55] Xiaoxuan Yang, Bonan Yan, Hai Li, and Yiran Chen. 2020. ReTransformer: ReRAM-based processing-in-memory architecture for transformer acceleration. In *Proceedings of the 39th International Conference on Computer-Aided Design*. 1–9.
- [56] Donghyeon Yi, Seoyoung Lee, Injun Choi, Gichan Yun, Edward Jongyoon Choi, Jonghee Park, Jonghoon Kwak, Sung-Joon Jang, Sohyun Ha, Ik-Joon Chang, et al. 2024. Skew-CIM: Process-Variation-Resilient and Energy-Efficient Computation-in-Memory Design Technique With Skewed Weights. *IEEE Transactions on Circuits and Systems I: Regular Papers* (2024).
- [57] Shihui Yin, Zhewei Jiang, Jae-Sun Seo, and Mingoo Seok. 2020. XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks. *IEEE Journal of Solid-State Circuits* 55, 6 (2020), 1733–1743.
- [58] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. Transpim: A memory-based acceleration via software-hardware co-design for transformer. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1071–1085.
- [59] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).