Scalable Sparse Transformer Accelerator with In-Memory Butterfly Zero Skipper and Local Attention Reusable Engine for Irregular-Pruned NN

Shiwei Liu, Jiangnan Yu, Peizhe Li, Feng Lin, Chixiao Chen, *Member, IEEE*

***Abstract*—Transformer-based large language models (LLMs) have achieved unprecedented advances across diverse AI tasks. However, their execution remains power-hungry, primarily due to the rapidly growing model size and the quadratic complexity of the self-attention. In-memory computing (IMC) has emerged as a promising solution for accelerating LLMs, delivering high storage density, wide memory bandwidth, and intrinsic compute parallelism. To further improve energy efficiency, this paper addresses LLM inference by leveraging irregular sparse computing in digital SRAM-based IMCs. The proposed butterfly-network-based feed-forward IMC engine enables irregular zero-weight skipping while sustaining high compute utilization. In parallel, a local-attention-reusable IMC engine supports variable sparse attention spans and eliminates redundant self-attention operations. Finally, we implement a four-chip system to scale the parallel performance of the IMC chip. Extensive experiments are deployed on the system, including BERT, GPT to LLAMA models on 11 AI tasks. The results show that the fabricated accelerator in 28nm CMOS achieves a peak throughput of 186.9 tokens/s and an energy cost of 0.21 J/token, yielding 4.2× speedup and 27.6× energy efficiency over the A100 GPU. Compared to the existing LLM-specific accelerators, the proposed accelerator further achieves 1.2× to 16.9× energy efficiency.**

***Index Terms*—Large Language Model, Sparse Transformer, In-Memory Computing, Butterfly Network, Reusable Local Attention**

I. Introduction

TRANSFORM-based LLM models have demonstrated unprecedented advances in a variety of AI tasks, including natural language processing [1], [2], computer vision [3], [4] and bioinformatics [5], [6]. A key advancement of LLMs lies in their superior capability to capture long-range contextual dependencies, enabled by remarkable parameter scalability and the self-attention mechanism. Over the five-year evolution from GPT-2 [7] to GPT-4 [8], model parameters grew from 1.5 billion to 1.7 trillion, while the maximum context length increased from 1K to 128K tokens. As LLMs continue to expand, there is a simultaneous need to scale compute throughput, memory capacity, and inter-chip bandwidth.

Fig. 1 depicts the typical structure and computing flow of transformers, which mainly consist of stacked self-attention and feed-forward layers. Queries (Q), keys (K) and values (V) are first obtained by multiplying input tokens with three weight matrices. Afterward, scores (S) that evaluate $Q\times K$ relevance are computed as scaled dot products and converted to probabilities (P) through the softmax function. Then, the $P\times V$ generates self-attention results. Finally, the self-attention outputs are fed into the feed-forward layer, which comprises several linear layers separated by non-linear activation functions. Matrix multiplication constitutes the primary workload in LLMs, and GPUs are regarded as the standard platform for their acceleration. However, GPUs are often constrained by the limited bandwidth between memory and processor chips [9], as large-scale weights and KV caches must be frequently transferred.

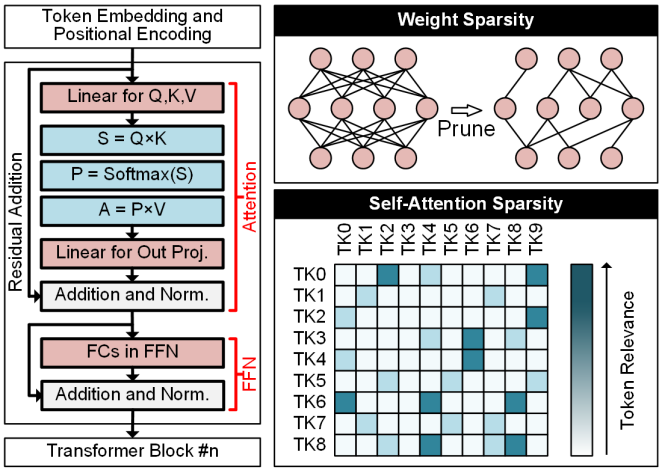


Fig. 1. Transformer based LLM algorithm and sparsity.

In contrast, in-memory computing (IMC) offers a promising solution for memory-intensive LLMs. By tightly coupling compute logic with storage cells, IMCs circumvent the performance bottleneck imposed by limited memory–processor bandwidth. Pioneering designs leverage workload partitioning [10], [11], parallelization strategies [12], [13], and model compression [14], [15], [16] to accelerate either linear layers or self-attention in transformers. However, achieving end-to-end efficiency improvements requires comprehensive optimization across both components. For example, the LLAMA-7B model [17] with 16K tokens requires 6.4 billion weights for its linear layers and 4 billion for the KV caches in self-attention. Consequently, efficient execution of LLMs demands a holistic IMC solution that accelerates both linear layers and self-attention.

Shiwei Liu is with the Shaoxin Laboratory, Zhejiang, China

Peizhe Li, Feng Lin, Chixiao Chen are with the State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China. Chixiao Chen is also with the Shaoxin Laboratory, Zhejiang, China

Jiangnan Yu is with the Hong Kong University of Science and Technology, Hong Kong SAR, China

Chixiao Chen is the corresponding author. (E-mail: cxchen@fudan.edu.cn)

Sparsification Techniques can boost the efficiency. As shown in Fig 1, weight sparsity strategically prunes useless weights while preserving model accuracy. Nevertheless, weight sparsity introduces irregular memory access and computation patterns, which are inherently incompatible with the regular IMC architectures. Therefore, previous IMC designs have adopted block-wise [18], [19], [20] or channel-wise [21], [22] sparsity patterns to maintain IMC utilization. However, structured sparsity restricts the distribution of non-zero weights (NZWs), which can lead to the unintended pruning of essential weights and potentially degrade model accuracy [23]. Unlike weight sparsity, self-attention sparsity arises from token relevance. Specifically, self-attention primarily allocates contextual information among strongly related tokens with a higher $Q{\times}K$ relevance score. The interactions between weakly related tokens can be skipped. Similarly, executing self-attention sparsity in IMCs also faces a trade-off between hardware efficiency and LLM accuracy.

In summary, efficient LLM execution in IMCs requires satisfying the aforementioned requirements: ❶ the IMC accelerator should provide comprehensive optimization across the full LLM stack, encompassing both linear layer and self-attention. ❷ the IMC accelerator should accommodate irregular sparsity to minimize the memory footprint without compromising model accuracy and hardware utilization. ❸ Finally, the IMC accelerator should demonstrate good scalability to support future LLMs. To address these challenges, this paper presents an SRAM-based full-digital-IMC transformer accelerator, featuring custom in-memory data distribution to handle both irregular weight sparsity and self-attention sparsity. The key contributions are:

- To handle the irregular weight sparsity, we propose a butterfly-network feed-forward IMC engine (BFLY-Engine) that enables irregular zero-weight skipping while maintaining high IMC utilization. A sparse-to-dense routing algorithm extracts activations to be computed with the corresponding NZWs. In addition, a transmission-gate butterfly network circuit enables efficient in-memory routing with minimal power consumption.
- To handle the irregular self-attention sparsity, we propose a local-attention-reusable IMC engine (LAR-Engine) to facilitate variable sparse attention spans. Combined with the QK sharing algorithm [24], the LAR-Engine further improves attention-span reuse and eliminates redundant self-attention operations.
- A silicon prototype of the proposed accelerator is fabricated in 28nm CMOS technology, and a four-chip system is assembled to enable end-to-end LLM inference and validate its scalability. The measurement results show that the multi-chip system achieves a high utilization with more devices. In addition, the system achieves a peak throughput of 186.9 tokens/s and an energy cost of 0.21 J/token, yielding 4.2× speedup and 27.6× energy efficiency over the Nvidia A100 GPU. Compared to state-of-the-art LLM accelerators, the proposed design further achieves 1.2× to 16.9× energy efficiency.

This paper is an extension of [25], where emerging topics, such as accelerator scalability, are further investigated. It also elaborates on the implementation details and provides more experimental results, demonstrating the effectiveness of the proposed design on modern LLM models. The rest of the article is organized as follows: Section II presents background and motivation. Section III describes the overall architecture. The proposed BFLY-Engine and LAR-Engine are explained in Section IV and Section V, respectively. Experimental results are demonstrated in Section VI, and Section VII concludes this article.

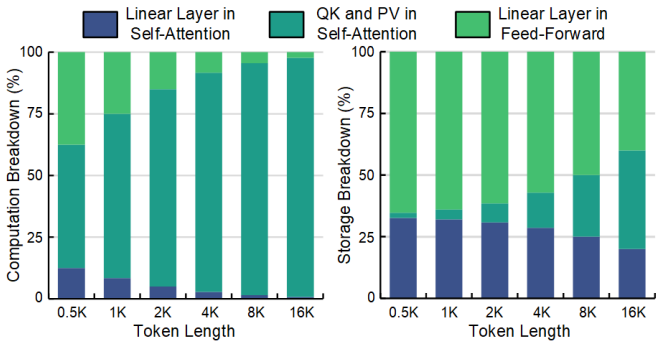


Fig. 2. (a) Computation and (b) Storage breakdown of various token length in LLAMA-7B model.

II. Background and Motivation

A. LLM Preliminaries

Parameter scalability and self-attention are fundamental to the effectiveness of LLMs. Scaling up the parameter size increases representational capacity, enabling richer feature extraction and better generalization. Meanwhile, the self-attention enhances performance by capturing long-range dependencies and focusing on relevant tokens, enabling LLMs to model complex language patterns. However, these two factors are both the maker and breaker of LLM's success. LLM parameters increase roughly 410× every two years, imposing substantial memory overhead [9]. In addition, self-attention conducts $Q{\times}K$ and $P{\times}V$ to each token pair, resulting in quadratic computation and memory costs with respect to token length.

Various accelerators have been proposed to improve LLM efficiency. Early approaches exploit attention sparsity [26], [27], KV cache management [28], and efficient self-attention variants [29], [30] to reduce the complexity of self-attention, while later works leverage mixed-precision techniques [31], [32] and parallelization strategies [33], [34] to mitigate parameter overhead. However, most of these works emphasize one-sided optimization, making it difficult to achieve end-to-end efficiency. Fig. 2 shows the computation and storage breakdown of the LLAMA-7B model for various token lengths. When token length <0.5K, linear layers dominate LLM computation. For longer sequences (>8K tokens), the $Q{\times}K$ and $P{\times}V$ in self-attention account for over 90% of the computation workload. A similar trend is observed in the storage breakdown. Thus, the LLM bottleneck shifts between linear layers and self-attention depending on the application.

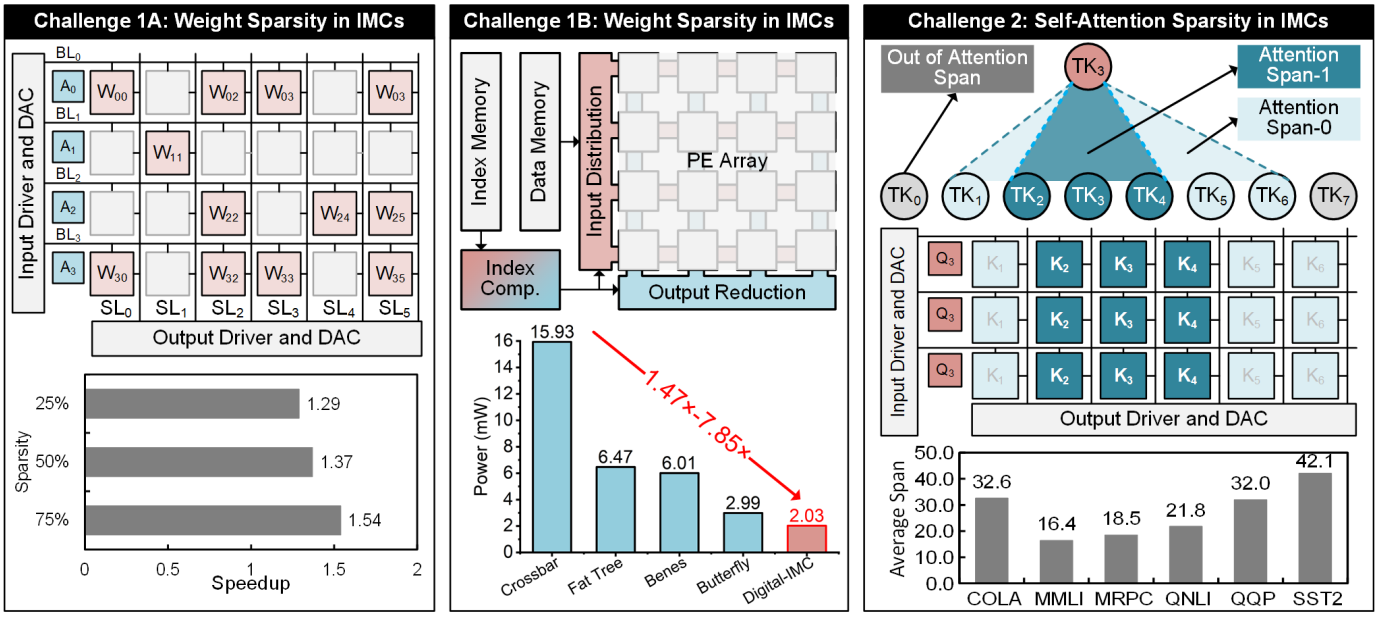


Fig. 3. Challenge of mapping sparse LLMs to IMCs.

B. Motivation of Weight Sparsity in IMC

Advanced sparsification algorithms [35], [36] can prune redundant weights while preserving LLM accuracy. But realizing these potential software benefits in hardware is challenging, particularly for IMC designs. As shown in Fig. 3, conventional IMCs are organized as crossbars, with memory arrays storing the weight matrix. Input activations are fetched from the interface and broadcast along the bitlines to all weights in the row. The resulting products are accumulated along the source lines to produce the final output. This crossbar-style IMC provides high storage density, wide memory bandwidth, and inherent compute parallelism. However, the rigid structure also prevents IMCs from efficiently mapping sparse weight matrices. As shown in Fig. 3, zero weights still occupy memory cells to maintain processing synchronization, leading to IMC underutilization. Even with only 25% of weights preserved (75% sparsity), directly mapping a 4K×4K sparse matrix to a 32×32 IMC crossbar yields just a 1.54× speedup. Previous IMC works prune weight matrices at the block granularity to fit preserved weight blocks into IMC crossbars. However, PatDNN [23] proves that such structured sparsity can cause significant accuracy degradation. Therefore, ❶ ***Challenge 1A*** **is how to map sparse weight matrices onto IMCs while preserving both model accuracy and IMC utilization.**

Accelerating sparse computing has been extensively studied in conventional von Neumann architectures [37], [38], [39]. As shown in Fig. 3, the representative Sigma [37] adopts an input distribution network and an output reduction network between the compute and memory subsystems. These interconnections configure the datapath and schedule the computation order of sparse matrices to achieve high hardware utilization. However, directly embedding distribution networks into compact IMC circuits incurs power overhead. We synthesize kinds of distribution networks at 200MHz and 0.9V in 28nm CMOS. The results in Fig. 3 show that the distribution networks consume 1.47× to 7.85× more power, compared to the SRAM-IMC used in our design (detailed in Section III). Thus, ❷ ***Challenge 1B*** **is how to realize efficient in-memory routing within IMC designs**.

To address these two challenges, we propose a butterfly-network-based feed-forward IMC engine in Section. IV. A sparse-to-dense routing algorithm enables irregular zero-weight skipping while sustaining high IMC utilization. A transmission-gate butterfly network circuit provides efficient in-memory routing with minimal power consumption.

C. Motivation of Self-Attention Sparsity in IMC

The quadratic complexity of self-attention prevents LLMs from handling longer sequences. Fortunately, Google and OpenAI have demonstrated the predominance of short-range token dependencies in LLMs. Their most advanced models, such as Gemini [40] and GPT-OSS [41], employ local attention to reduce the quadratic complexity to linear, where each token attends only to a fixed window of neighboring tokens. This window size is referred to the attention span. Moreover, we further observe that the optimal attention span can vary across different models, and even across individual layers, depending on the task. We prune self-attention in the BERT-110M model to local attention with a 1% accuracy loss budget on six GLUT tasks. The default self-attention span is 128. As shown in Fig. 3, the GPT model exhibits average attention spans ranging from 16.4 to 42.1 tokens across different tasks. Similarly, due to the fixed size of IMC arrays, conventional IMC designs cannot accommodate varying attention spans. For example, attention span-0 in Fig. 3 can fully utilize the IMC array, whereas span-1 achieves only 50% utilization. Accordingly, ❸ ***Challenge 2*** **is how to support varying local attentions within IMC designs**.

To address this challenge, we propose a local-attention-reusable IMC engine that enhances the reuse of local atten-

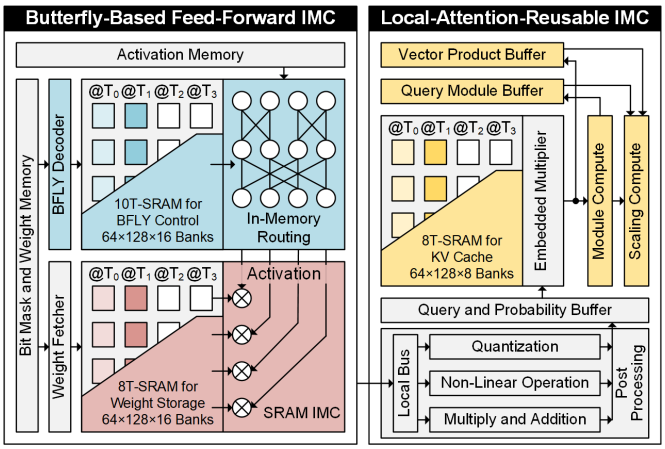


Fig. 4. Overall architecture of the proposed sparse LLM accelerator.

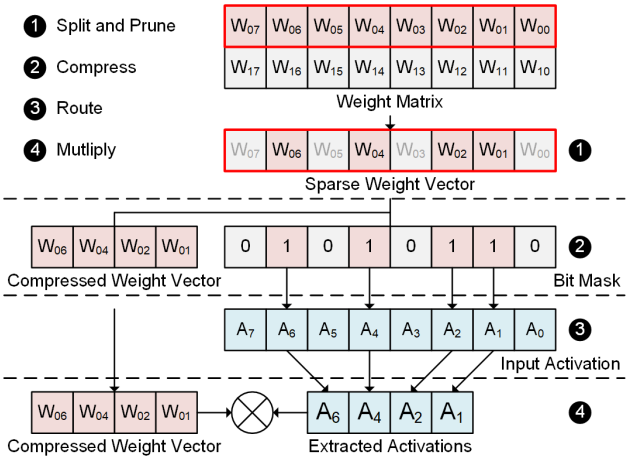


Fig. 5. Sparse-to-dense feed-forward flow.

tion span. Combined with QK sharing, it further eliminates redundant self-attention operations. The implementation is described in Section V. Most importantly, we fabricate a four-chip system to combine the BFLY-Engine and LAR-Engine to enable end-to-end LLM inference and accelerator scalability.

III. Overall Architecture

Fig. 4 shows the overall architecture of a single transformer core in the proposed sparse LLM accelerator. The accelerator comprises four such cores, each including a BFLY-Engine for linear layer computation and a LAR-Engine for self-attention computation. The BFLY-Engine is further divided into an in-memory BFLY-routing macro and an SRAM-IMC macro. The routing macro distributes activations to NZWs according to their distribution, while the IMC macro receives these extracted activations and performs multiplication-and-accumulation (MAC) with the compressed weight matrix. To reduce bitline capacitance, the routing macro and IMC macro are divided into 16 banks, sized 64 bits $\times$ 128 depth. The LAR-Engine consists of an SRAM-based IMC macro and a post-processing macro. Similarly, the IMC macro is partitioned into eight 64-bit $\times$ 128-depth banks along with a shared buffer group. The IMC macro stores the KV cache and generates $Q\times K$ and $P\times V$ results. The vector product buffer and query module buffer hold the $Q\times K$ results for reuse, preventing redundant self-attention operations. Finally, the post-processing macro applies activation functions, including softmax, normalization, and residual addition.

To fully accommodate LLMs, the accelerator operates in three stages. In the first stage, the BFLY-Engine sequentially generates Q and K, while the LAR-Engine performs $Q\times K$ in pipeline. The post processing then converts the scores into probabilities. Following $Q\times K$ is $P\times V$. The BFLY-Engine and LAR-Engine also operate in the pipeline mode, to generate V and $P\times V$, respectively. Finally, the results of multi-head attention are concatenated in the activation memory as input for subsequent linear layers. Both the BFLY-Engine and LAR-Engine operate in parallel mode to process the feed-forward linear layers concurrently.

IV. Butterfly-Based Sparse Feed-forward IMC

This section addresses *Challenge 1A* and *Challenge 1B* outlined in Section II, with a focus on reducing the memory footprint of linear layers in LLMs. We first present a sparse-to-dense routing algorithm that enables irregular zero-weight skipping while preserving high IMC utilization. Next, we introduce a transmission-gate butterfly circuit to realize efficient in-memory activation distribution.

A. Sparse-to-Dense Feed-Forward

1) ***Sparse-to-Dense Computing Flow****:* Fig. 5 illustrates a walk-through example of the sparse-to-dense feed-forward flow. First, the pretrained weight matrix is divided into multiple equal-sized weight vectors, each pruned with a unified sparsity ratio. Notably, the preserved NZWs are randomly distributed within the weight vector. Second, the sparse weight vector is compressed into a dense vector along with a binary-encoded bitmask, where bit-1 marks the relative position of NZWs in the original weight matrix. Finally, the required activations are extracted from the original inputs and routed to their paired NZWs based on the bitmask. The vector product is then obtained by multiplying the two resulting dense vectors.

This irregular $N{:}M$ weight pruning method has been widely adopted for LLM sparsification [42], [43], [44]. Here, N denotes the size of the split weight vector, and M represents the number of preserved NZWs. The $N{:}M$ pruning bridges the performance gap between the two extremes of fully unstructured pruning and fully structured pruning. The local fine-grained sparsity prevents the erroneous pruning of important weights, while global coarse-grained sparsity can be leveraged to achieve potential hardware benefits.

2) ***Butterfly Network Decoding****:* The key challenge of the sparse-to-dense feed-forward flow is routing activations to NZWs according to weight bitmasks. While the multiplexer (MUX) can easily perform this function, its hardware cost grows exponentially with large N and M. To overcome this limitation, we employ a butterfly-type data distribution network along with a corresponding routing algorithm to efficiently distribute activations. As shown in Fig. 6, the butterfly network has a multistage structure composed of log_2N stages

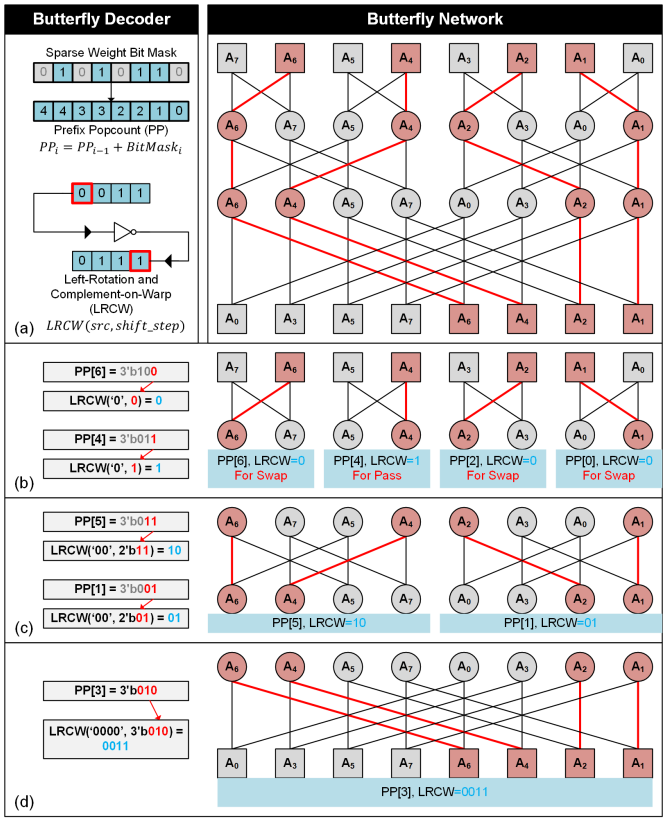


Fig. 6. (a) Butterfly network and decoding logic, and (b)-(d) decode butterfly network from level-0 to level-2.

for N entries. Each stage consists of $N/2$ switches, where each pair of activations is either swapped or passed through to the next stage.

Fig. 6(a)-(d) shows an example of the decoding procedure for an 8-entry butterfly network. The decoding logic converts NZWs' bitmasks into control bits to configure the butterfly routing path. The required activations, highlighted in red in Fig. 6(a), are routed to the bottom-right while preserving their original order in the activation vector, ensuring correct computation. The decoding logic consists of two primary operations: prefix popcount (PP) and left-rotation complement-on-wrap (LRCW). The popcount computes the cumulative sum of a given bitmask, where the count of bit-1 indicates the number of NZWs up to a certain position. Meanwhile, the LRCW performs a standard left rotation, except that the bit is complemented whenever it wraps around. For the first stage in Fig. 6(b), the least significant bit (LSB) of popcount results, *PP[6,4,2,0]*, are used as rotation steps, which are then applied to execute four parallel LRCWs on a zero sequence ('0'). This generates control bits of '0, 1, 0, 0'. If a control bit is 1, the input activation pair passes through the butterfly switch. Otherwise, the inputs are swapped. In the second stage, shown in Fig. 6(c), the two LSBs of *PP[5,1]* are used as rotation steps to perform two parallel LRCWs on the zero sequence ('00'), producing control bits '10' and '01'. Moving to the final stage in Fig. 6(d), three LSBs of *PP[3]* rotate the four-zero sequence ('0000'), producing control bits '0011'. This decoding method transforms the irregular *N:M* spare computing to the dense vector product, which could be easily accelerated by IMCs.

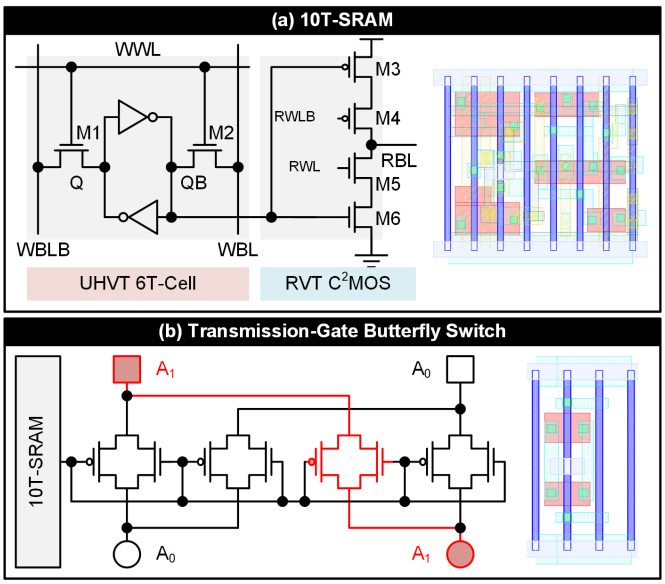


Fig. 7. The circuit and layout of (a) 10T SRAM, and (b) butterfly switch

B. In-memory Butterfly Network Implementation

In practice, we implement a 32:8 butterfly network to support the sparse-to-dense feed-forward computing. The first three stages of the 32-entry butterfly network are partitioned into four identical 8-entry sub-networks, each configured with distinct control bits. The final two stages are simplified into 4:1 MUXs, that select outputs from the sub-networks to generate the extracted activations. Both the sub-networks and MUXs follow the same decoding method illustrated in Fig 6. Ideally, the 32:8 butterfly network achieves peak efficiency at 75% weight sparsity. To support a higher NZW ratio, such as 50% sparsity, the network selects eight non-skipped activations from 16 inputs while gating the unused switches. Conversely, for 87.5% sparsity, half of the subsequent IMC array is gated.

To reduce the hardware cost of embedding the butterfly network into the IMC array, we adopt a custom in-memory design that employs 10T-SRAM cells for storing the generated control bits, and cascaded transmission gates (TGs) to implement the butterfly network. Fig. 7 shows the circuit and layout of the 10T-SRAM and TG-based butterfly switch. The 10T-SRAM combines a 6T-SRAM cell with ultra-high threshold (UHVT) transistors to reduce leakage power and a regular-threshold (RVT) C^{2}MOS gate. The combinational C^{2}MOS allows direct readout, eliminating the need for a sequential sense amplifier as in standard 6T-SRAM. In addition, compared to the 8T-SRAM used in our IMC design, the C^{2}MOS further removes the read bitline (RBL) precharge, enabling high-frequency activation distribution. As shown in Fig. 8(c), the proposed 10T-SRAM reduces dynamic power by 1.22× and 3.31× compared to the 6T and 8T cells, respectively. Each butterfly switch consists of four TGs, controlled by the RBL of a single 10T-SRAM cell. All switches are directly connected without additional buffers to minimize hardware cost. Finally, the butterfly decoding logic is synthesized using digital standard cells, with the prefix popcount implemented via cascaded 1-bit adders and the LRCW realized using a barrel shifter. The IMC employed in the BFLY-Engine is the same as that used

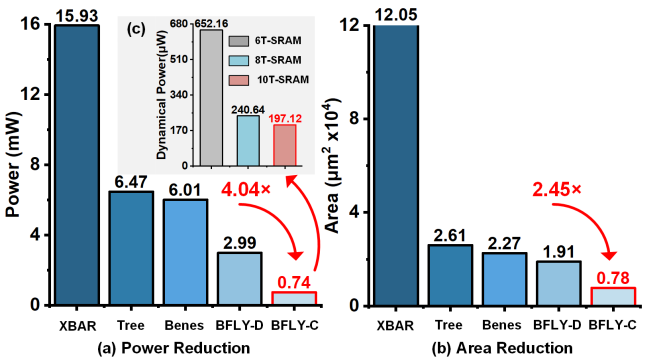


Fig. 8. (a) power and (b) area costs reduction of the butterfly network, and (c) power reduction of the 10T SRAM.

in the LAR-Engine, which will be explained in Section V.

C. Evaluation

Fig.8 shows the evaluation result of the custom in-memory butterfly network (BFLY-C). The baselines include various distribution networks implemented by conventional digital VLSI methodologies (synthesis, placement and routing). All designs support the same data distribution range and operate at 200MHz and 0.9V. Compared to the XBAR, Fat Tree and Benes networks, the butterfly network exhibits the lowest complexity of $N/2 \cdot log_2 N$. In addition, the use of TG-based switches further reduces the circuit implementation cost. As a result, the BFLY-C achieves 8.1× to 21.5× lower power consumption and 2.9× to 15.4× smaller area. Compared to the digital butterfly counterpart (BFLY-D in Fig.8), our custom in-memory design further reduces power consumption by 4.0× and area by 2.5×. The impact of the BFLY-Engine on the overall accelerator performance will be discussed in Section VI.

V. Local-Attention-Reusable IMC

This section tackles *Challenge 2* outlined in Section II by reducing the quadratic complexity of the standard self-attention to linear. The proposed LAR-Engine enhances reuse in local attention. Combined with *QK* sharing, the LAR-Engine further eliminates redundant self-attention operations.

A. Reusable Sparse Local Attention

1) ***Local Attention and QK Sharing****:* The standard self-attention computes the vector product for every token pair, thus resulting in quadratic complexity with respect to token length. To reduce this complexity, we employ two efficient self-attention variants, local attention and *QK* sharing, as illustrated in Fig. 9. Local attention restricts each token to attend only to a span of nearby tokens. This reduces complexity from $O(n^2)$ to $O(n \cdot S)$, which is linear in token length when $n \ll S$. Here n is the token length and S denotes the size of the local attention span. On the other hand, *QK* sharing sets the K as the normalized Q, avoiding both K-related storage and computation. In addition, the $Q \cdot K$ matrix becomes nearly symmetric, except for the Q module. For example, as shown in Fig. 9, the vector products $Q_1 \times K_3$ and $Q_3 \times K_1$ share a

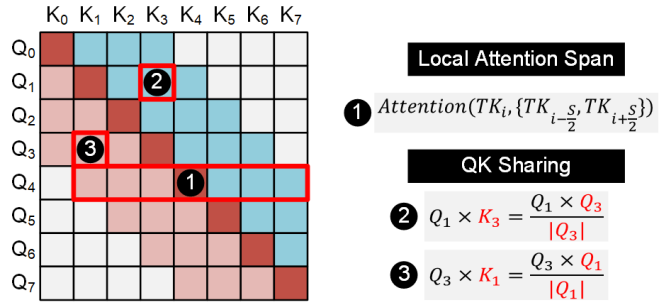


Fig. 9. Sparse self-attention with local attention span and QK sharing.

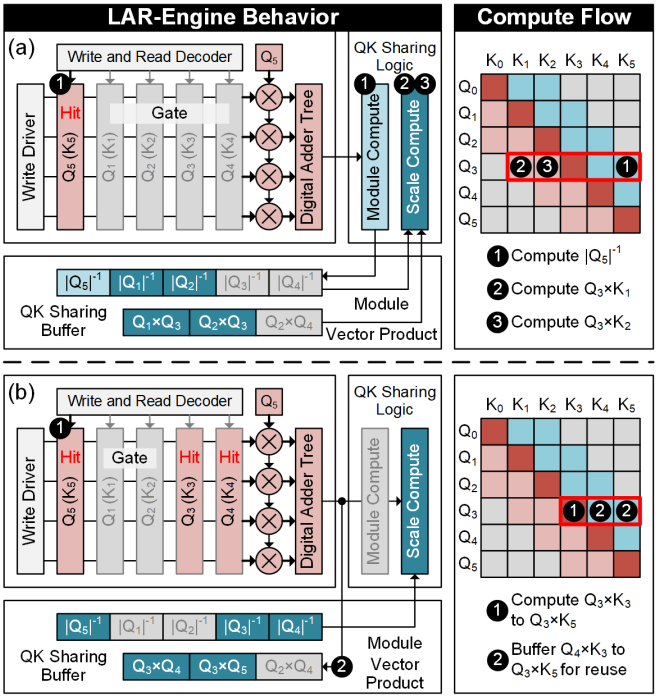


Fig. 10. Walk-through example of executing local attention with QK sharing on the LAR-Engine.

common $Q_1 \times Q_3$ but involve different Q_1 and Q_3 modules. Notably, *QK* sharing applies only to transformer encoders, like BERT. In the auto-regressive transformer decoder, each token attends only to previous tokens, making the $Q \cdot K$ matrix in Fig. 9 lower triangular. Even so, encoders are widely used in modern LLMs, such as OpenAI SORA [3].

2) ***Local Attention Reuse****:* The proposed LAR-Engine translates the underlying twofold reuse in both local attention and *QK* sharing to tangible hardware gains. As shown in Fig. 10, we take the computation of Q_3 as a walk-through example to present the processing flow.

In this example, Q_3 should attend to K_1 through K_5. Fig. 10(a) first illustrates the computation of $Q_3 \times K_1$ and $Q_3 \times K_2$. For reuse in local attention, only one K differs between two adjacent attention spans. Therefore, the popped-out K_0 is replaced by the pushed-in K_5. If *QK* sharing is applied, the IMC array updates Q_0 with Q_5, while the embedded multiplier in IMC and module unit computes the module of Q_5. Meanwhile, previous Q_1 to Q_4 remain stationary for future reuse. For reuse in *QK* sharing, the result of $Q_3 \times K_1$ and $Q_3 \times K_2$ can be retrieved from module buffer and vector-

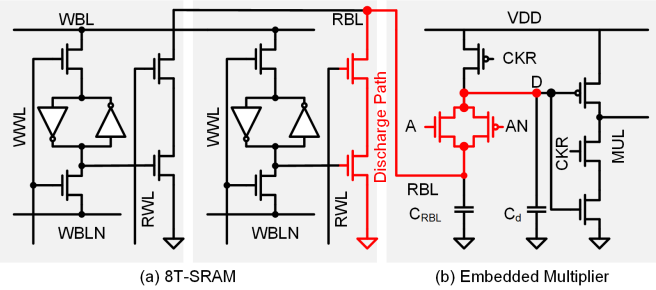


Fig. 11. The circuit diagram of (a) 8T-SRAM, and (b) embedded dynamic multiplier.

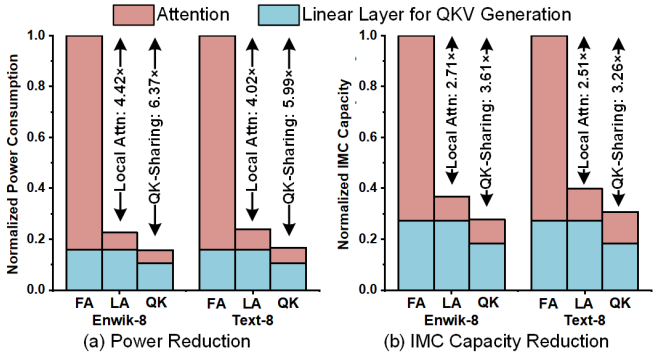


Fig. 12. LAR-Engine evaluation results: (a) Power, and (b) IMC capacity reduction. FA refers to standard self-attention, LA refers to local attention, and QK is local attention with QK sharing.

product buffer. Specifically, the previous intermediate results of $Q_1 \times Q_3$ and $Q_2 \times Q_3$ are combined with modules of Q_1 and Q_2, while the IMC array is gated for power saving. Thus, the LAR-Engine leverages QK sharing to reuse self-attention computation and eliminate redundant operations.

Fig. 10(b) illustrates the computation between Q_3 and K_3-K_5. With *QK* sharing, the IMC array performs vector products of Q_3 with Q_3-Q_5, and the module buffer retrieves their corresponding modules. The results $Q_3 \times Q_4$ and $Q_3 \times Q_5$ are stored in the vector-product buffer for the subsequent reuse.

B. LAR-Engine Implementation

Fig 10 already shows the overall IMC architecture of the LAR-Engine, which is also adopted by the BFLY-Engine. Departing from conventional IMC crossbars, the proposed design places embedded multipliers along the boundaries of the SRAM banks. This computing-on-boundary IMC architecture provides greater flexibility for managing *QK* updates in the LAR-Engine, and for integrating the in-memory butterfly network in the BFLY-Engine. The accelerator leverages multiple such IMCs to improve computational parallelism.

Fig. 11 illustrates the circuit of a single row in the computing-on-boundary IMC. The 8T-SRAM cells are connected through the RBL to the embedded multiplier. The embedded multiplier is implemented as a 1-bit dynamic AND gate, building on our previous work [45], which integrates the readout circuit, multiplier, and dynamic latch into a single functional unit. The dynamic AND gate leverages bit-wise sparsity to reduce dynamic switching power on the RBL. Specifically, in each MAC operation, CKR is first set low

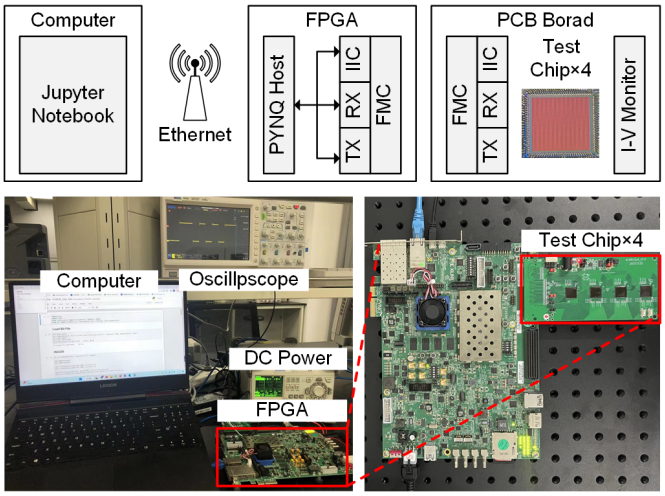


Fig. 13. Test platform.

to precharge latch D, then raised high to start the operation. The 8T-SRAM stores the key (or weight), while the query (or activation) is applied to the A/AN port. The path from latch D to the RBL discharges only when both query and key are bit-1, producing a bit-1 output. If either input is zero, the RBL discharge is avoided, thereby exploiting bitwise sparsity without requiring any explicit zero-detection logic. Finally, in the LAR-Engine, the query module buffer is a 16-bit, 128-depth SRAM, while the vector-product buffer is a 16-bit, 64-depth SRAM, supporting local attention and QK sharing for on-chip attention spans of 128. The non-linear operations, including module computation and softmax are implemented using a 256-entry lookup table for INT-8 precision.

C. Evaluation

Fig. 12 shows the evaluation result of LAR-Engine with local attention and *QK* sharing. Experiments are conducted using GPT2-124M with a default token length of 1K. With only local attention, the LAR-Engine reduces the computation and storage overhead for the *KV* cache, achieving 4.42× and 4.02× lower power, as well as 2.71× and 2.51× smaller IMC capacity on the Enwik-8 and Text-8 datasets, respectively. *QK* sharing further eliminates the linear computation for *K* generation and redundant $Q \times K$ operations in self-attention. As a result, power consumption is reduced by 6.37× and 5.99×, while IMC capacity is reduced by 3.61× and 3.26× on the Enwik-8 and Text-8 datasets, respectively.

VI. Measurement Result

A. Test Platform Setup

Fig 13 gives the test platform for the proposed accelerator, which is fabricated in 28nm CMOS. The FPGA serves as off-chip memory to store model weights, *KV* cache, and other intermediate results. Data and instructions from the host computer are transmitted to the test chip through the FPGA mezzanine connector (FMC). The test PCB board carries voltage and current monitors for power measurement and hosts four test chips. The test chip performs the main transformer computations, including self-attention and feed-forward layers,

TABLE I
BERT-110M PERFORMANCE ON GLUE AND SQUAD DATASETS

Tasks		COLA	MNLI	MPRC	QNLI	QQP	RTE	SST2	SQUAD
Metric		Matthews Correlation	Accuracy	F1 Score	Accuracy	Accuracy	Accuracy	Accuracy	F1 Score
BERT-110M	Baseline Accuracy	54.2	83.1	90.2	90.7	90.6	67.5	90.6	88.6
	Sparse Accuracy	53.1	81.7	88.4	88.9	89.1	63.1	89.8	85.8
	Attention Span	32.6	16.4	18.5	21.8	32.0	47.3	42.1	152.3

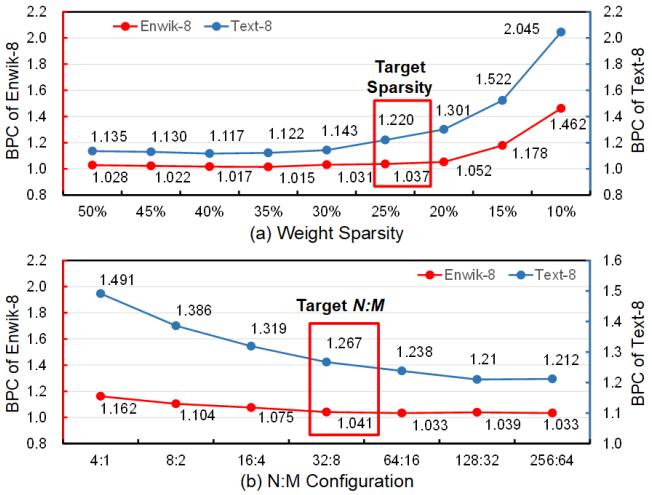


Fig. 14. The BPC performance of GPT2-124M on the Enwik-8 and Text-8 datasets under (a) varying sparsity with unstructured pruning and (b) 50% sparsity with N:M pruning.

while preprocessing operations such as token embedding and positional encoding are handled by the host computer. As shown in Fig. 17, the four chips, together with the FPGA, form a scalable system that enables end-to-end LLM inference.

We evaluate the test chip on two smaller transformer models, BERT-110M and GPT2-124M, as well as a large-scale generative LLM model, LLAMA-7B. For BERT, performance is assessed on seven text classification tasks from the GLUE datase and the question-answering task from the SQUAD v1.1 dataset. For GPT2 and LLAMA, performance is evaluated on three text generation tasks using the WikiText-2, Enwik-8, and Text-8 datasets. We prune the weights using Wanda [35] and the self-attention using Adaptive-Span [46], followed by quantizing the sparse models to INT8 with SmoothQuant [47]. Finally, we finetune the resulting models to recover performance.

B. LLM Accuracy

We first determine the size of the in-memory butterfly network for weight sparsity. To this end, we evaluate the bits-per-character (BPC) metric of GPT2-124M model on the Enwik-8 and Text-8 datasets, where a lower BPC indicates better performance. Fig 14(a) shows the BPC of the GPT model pruned with fully unstructured sparsity. The model maintains its performance until 75% of the weights are pruned. Therefore, a 75% sparsity ratio is chosen as the target for the subsequent *N:M* pruning. Then, we exploit an appropriate *N:M* configuration to preserve the BPC performance under

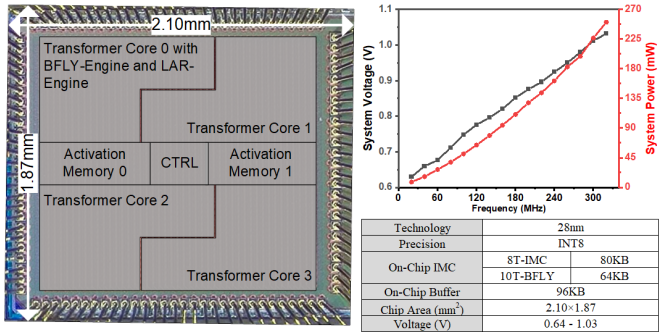
TABLE II
GPT2-124M AND LLAMA-7B PERFORMANCE ON WIKITEXT-2, ENWIK-8 AND TEXT-8 DATASETS

Tasks		WikiText-2	Enwik-8	Text-8
Metric		PPL	BPC	BPB
GPT2-124M	Baseline Accuracy	29.40	1.04	1.26
	Sparse Accuracy	32.47	1.07	1.42
	Attention Span	115.3	83.3	97.9
LLAMA-7B	Baseline Accuracy	5.47	0.72	0.81
	Sparse Accuracy	11.79	0.99	1.08
	Attention Span	256		

75% sparsity. A larger *N* imposes fewer constraints on the distribution of NZWs, theoretically yielding better BPC values, but requires a larger butterfly network. As shown in Fig. 14(b), the GPT model on Enwik-8 incurs only a 0.04 BPC loss at *N:M* = 16:4, while the GPT model on Text-8 experiences only a 0.05 BPC loss at *N:M* = 32:8. To balance the hardware efficiency and model performance, we conservatively choose *N:M* as 32:8. This configuration is adopted for the remaining experiments.

Table I presents the performance of BERT-110M on the GLUE and SQUAD datasets with both weight sparsity and self-attention sparsity. For tasks from the GLUE dataset, the default token length is 128. We report Matthews correlation for the COLA task, F1 score for the MRPC task, and accuracy for the remaining. For the SQUAD dataset, the token length is 512, and F1 score is used. Higher Matthews correlation, F1 score, or accuracy indicates better BERT performance. The sparse BERT achieves performance close to the dense baseline, with less than 2% loss on most tasks, except for the RTE task and the SQUAD dataset. The RTE task has a limited training set, which constrains finetuning and accuracy recovery. The SQUAD dataset is more challenging, making the model more sensitive to sparsity. Finally, Table I also gives the local attention span. BERT achieves 63.0% to 87.2% self-attention sparsity on the GLUE dataset and 70.3% on the SQuAD dataset, respectively.

Table II summarizes the performance of GPT2-124M and LLAMA-7B on the WikiText-2, Enwik-8 and Text-8 datasets. The input token length is set to 1K. For WikiText-2, we report perplexity (PPL), with lower values indicating better results, similar to BPC. In addition, *QK* sharing is not applied to these auto-regressive models. Compared to the baseline, the GPT model incurs a PPL loss of 3.07 and a BPC loss of 0.03/0.16 with 75% weight sparsity and 88.7% to 91.9% self-attention sparsity. The LLAMA model experiences a larger PPL loss



Technology	28nm	
Precision	INT8	
On-Chip IMC	8T-IMC	80KB
	10T-BFLY	64KB
On-Chip Buffer	96KB	
Chip Area (mm^2)	2.10×1.87	
Voltage (V)	0.64 - 1.03	
Frequency (MHz)	20 - 320	
Latency (ms)	BERT/GLUE[1]	111[3]/154[4]
	GPT2/Enwik[2]	841[3]/1086[4]
Peak Performance[2,3] (TOPS)	BFLY-Engine	1.31
	LAR-Engine	2.04
Energy Efficiency[2,3] (TOPS/W)	IMC Macro	2.52 - 32.17
	System	1.96-25.22
Area Efficiency[2,3] ($TOPS/mm^2$)	0.85	

1. BERT-110M on GLUE dataset: token length=128, weight sparsity=75%, local attention span=32;
2. GPT2-124M on Enwik-8 dataset, token length=1024, weight sparsity=75%, local attention span=83;
3. Exclude off-chip memory access;
4. Include off-chip memory access.

Fig. 15. Die photo, voltage-power-frequency scaling and specifications.

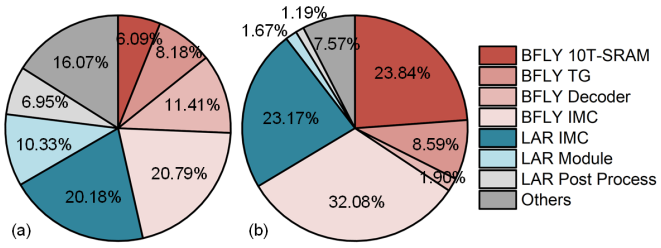


Fig. 16. (a) Power and (b) area breakdown.

of 6.32 and a BPC loss of 0.27, as it is overtrained and thus more difficult to finetune [48].

C. Chip Measurement Result

1) ***Chip Summary****:* Fig. 15 presents the die photo of the LLM accelerator, along with its voltage–power-frequency scaling and specifications. The accelerator is fabricated in 28nm CMOS technology, with 1.87×2.10mm^2 area. It operates at a power supply of 0.64–1.03V and a working frequency of 20–320MHz. To evaluate both IMC macro and system performance, power domains for custom BFLY-Engine, LAR-Engine and other synthesized digital circuits are separated. With 75% weight sparsity and 92% self-attention sparsity on GPT2-124M model and Enwik-8 dataset, the BFLY-Engine achieves a peak throughput of 1.31TOPS, while the LAR-Engine reaches a peak throughput of 2.04TOPS. For end-to-end GPT model inference, the combined BFLY-Engine and LAR-Engine IMC macros achieve the best energy efficiency of 32.17 TOPS/W at 0.66V and 40MHz. Overall, the accelerator achieves a peak energy efficiency of 25.22TOPS/W and an area efficiency of 0.85TOPS/mm^2.

Fig. 16 shows the power and area breakdown. The BFLY-Engine accounts for 46.46% of the power and 66.40% of the area. The LAR-Engine consumes 37.47% of the power and 26.03% of the area. The remaining is for the activation memory, chip interface and controller. Notably, the TG-based butterfly network occupies only 17.60% of the power and 12.93% of the area within the BFLY-Engine, enabling efficient in-memory activation distribution.

TABLE III
LATENCY OF BERT-110M ON SQUAD AND GPT2-124M ON ENWIK-8.

Model	Token Length	Operations	Compute Latency (ms)	DDR Access Latency (ms)	Overall Latency (ms)
BERT-110M	128	Linear Layer in Self-Attention	2.97	0.86	3.83
		QK and PV in Self-Attention	0.82	0.06	0.88
		Linear Layer in Feed-Forward	5.38	2.74	8.12
		Entire Model	110.04	43.92	153.96
GPT-124M	1024	Linear Layer in Self-Attention	16.72	4.83	21.55
		QK and PV in Self-Attention	18.29	1.52	19.81
		Linear Layer in Feed-Forward	34.76	14.37	49.13
		Entire Model	837.24	248.64	1085.88

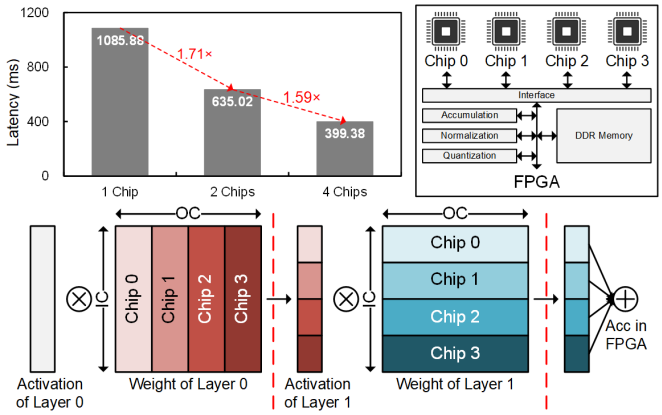


Fig. 17. Scalability of the proposed LLM accelerator on GPT2-124M model.

2) ***Processing Latency****:* Table III presents the layer-wise processing latency of BERT-110M and GPT2-124M inference on a single chip operating at 1.0V and 300MHz. This end-to-end latency includes off-chip DDR memory access in the FPGA, which provides a bandwidth of 12.8GB/s. We evaluate the BERT model on the GLUE dataset with an input token length of 128. For the GPT model, we use the Enwik-8 dataset with 892 input tokens and generate an additional 128 tokens. A 32:8 weight sparsity is adopted, while the BERT and GPT models employ local attention with average attention spans of 32.6 and 83.3, respectively. In the BERT model, the $Q{\times}K$, $P{\times}V$, and linear layers in self-attention consume 0.82ms and 2.97ms, respectively. The linear layer in the feed-forward further consumes 5.38ms. Due to the limited on-chip memory of the fabricated chip, off-chip memory access adds an additional 3.66ms to fetch weights, activations, and the *KV* cache. With 12 such transformer layers, the BERT model requires a total of 153.96ms to process 128 tokens. In contrast, with a longer token length, the $Q{\times}K$ and $P{\times}V$ account for a larger portion of GPT model inference, requiring 18.29ms. Overall, the GPT model takes 1085.88ms to complete a single inference.

3) ***Scalability****:* Fig. 17 illustrates the scalability of the proposed transformer accelerator for the GPT2-124M model when generating 128 tokens from 892 input tokens. The multi-chip system distributes self-attention across heads and linear

TABLE IV
COMPARISON WITH STATE-OF-THE-ART IMC-BASED LLM ACCELERATOR

	JSSC'23 [49]	JSSC'25 [50]	ESSERC'24 [51]	TCAS-I'25 [52]	JETCAS'25 [53]	**This Work**
Technology	28nm	28nm	65nm	22nm	20nm	28nm
Implementation	Digital SRAM IMC	Hybrid Analog SRAM-ROM IMC	Hybrd Digital-Analog SRAM IMC	Digital SRAM IMC	Digital DRAM IMC	Digital SRAM IMC
Application	BERT	LoRA	BERT	ViT	BERT, GPT	BERT, GPT, LLAMA
Precision	INT8	INT8	INT8	INT8	INT8	INT8
Die Area (mm2)	6.69	5.67	3.2	5.1	Not Report	3.93
Supply Voltage (V)	0.6 - 1.0	0.7 - 1.1	0.7 - 1.2	0.76 - 0.92	Not Report	0.64 - 1.03
Frequency (MHz)	80 - 240	120 - 220	350 - 1100	75 - 152	500MHz	20 - 320
Power (mW)	27.0 - 118.2	Not Report	49 - 456	17.1 - 48.3	Not Report	8.27 - 250.65
Peak Peformance (TOPS)	1.48	0.4	0.25	0.16	1.41[5]	0.49[6]/3.33[7,8]
Macro Energy Efficiency (TOPS/W)	Not Report	Not Report	14.8	Not Report	Not Report	BFLY-Engine: 1.76[6] - 15.60[7] LAR_Engine: 3.41[6] - 70.37[8]
System Energy Efficiency (TOPS/W)	20.5[1]	42.0[2]	1.65[3]	12.3[4]	1.485	25.22[7,8,9]
System Area Efficiency (TOPS/mm2)	0.22	0.16	0.08[3]	Not Report	Not Report	0.85[7,8,9]

1. Energy efficiency for global+local sparse self-attention, the sparsity is not reported; 2. Exclude ADC and peripheral circuits; 3. Energy efficiency for self-attention with 75% token pruning; 4. Energy efficiency for weight-activation multiplication with 74.6% weight sparsity; 5. Post-layout Simulation; 6. Without weight and self-attention sparsity; 7. With 75% weight sparsity; 8. With 92% self-attention sparsity; 9. GPT-124M model on single chip with 1K token.

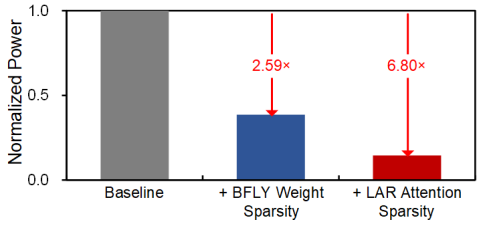


Fig. 18. Ablation analysis of BERT-110M model on SQUAD dataset with 512 tokens.

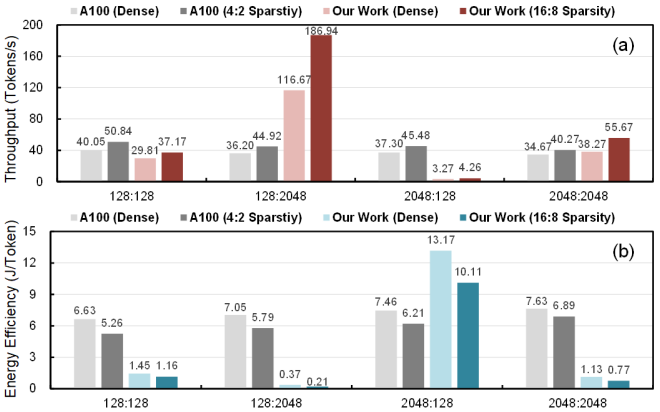


Fig. 19. (a) Throughput and (b) energy-efficiency improvements over Nvidia A100 GPU.

layers across channels. The first weight matrix is partitioned by output channels, allowing the resulting activations to be directly consumed by the second weight matrix. In contrast, the second weight matrix is partitioned by input channels. Finally, the partial sums are transmitted to the FPGA for synchronization. On average, the GPT inference requires 1085.88ms using a single chip, 635.02ms with two chips, and 399.38ms with four chips. The accelerator's performance scales up by 1.71× and 1.59×, indicating a high utilization with more devices. In addition, the performance gains are not strictly proportional to the number of devices, as adding more chips requires increased data synchronization.

*4) **Ablation Analysis**:* Fig. 18 shows the ablation study of the proposed in-memory sparsification for the BERT-110M model on the SQUAD dataset. The baseline is a comparable accelerator that follows the architecture in Fig. 4 but omits these sparsity optimizations. ❶ We first leverage weight sparsity in the BFLY-Engine. The *N:M* weight sparsity prunes 75% of the weights while preserving model accuracy. Additionally, the TG-based butterfly network reduces power consumption by 4.04× and area overhead by 2.45× for in-memory activation distribution. Consequently, these optimizations jointly reduce power consumption in the end-to-end BERT inference by 2.59×. ❷ We further exploit the sparse self-attention in the LAR-Engine. Local attention reduces the attention span from 512 to an average of 152.3, while QK sharing eliminates the need for key generation. The reusable local-attention flow converts the software benefit to hardware gains. It leverages on-boundary IMCs to avoid unnecessary QK updates, and enables computation reuse in the $Q \times K$ operation. Overall, the LAR-Engine, combined with the BFLY-Engine, achieves energy savings of 6.80×.

D. Comparison to State-of-the-Arts

*1) **Comparison to Nvidia A100 GPU**:* We first compare the proposed sparse LLM accelerator with the Nvidia A100 GPU. To provide more compute parallelism, we use eight chips to run the LLAMA-7B model. Since the FPGA cannot provide sufficient DDR interfaces for all eight chips simultaneously, we develop a cycle-level simulator assuming 12.8 GB/s bandwidth per device. In addition, to ensure a fair comparison at the same sparsity level, we implement 16:8 sparsity on the accelerator and 4:2 sparsity on the A100 GPU, which is inherently supported by Nvidia sparse tensor core. Even with fewer GPUs, the peak throughput of a single NVIDIA A100 GPU remains 47.3× higher than that of eight proposed chips. Fig. 19 shows the improvements in throughput and energy efficiency across different numbers of input (128/2048)

and output (2048/128) tokens. When processing 128 input tokens to generate an additional 2K tokens, the accelerator achieves a throughput of 186.9 tokens/s and an energy cost of 0.21 J/token, resulting in 4.2× speedup and 27.6× energy efficiency. In contrast, the accelerator exhibits lower performance when processing larger numbers of input tokens. Compared to the A100 GPU, it delivers 10.7× lower speedup and incurs 1.63× higher energy cost when using 2K input tokens to generate 128 tokens. That's because the compute-bound input token encoding can saturate GPU utilization, whereas the memory-bound output token decoding exacerbates the overhead caused by the back-and-forth data movement in GPUs.

*2) **Comparison to LLM Accelerators**:* We also compare the proposed sparse LLM accelerator with state-of-the-art designs [49], [50], [51], [52], [53] in Table IV. By jointly exploiting in-memory weight sparsity and self-attention sparsity, our work achieves the best energy efficiency of 25.22TOPS/W and area efficiency of 0.85TOPS/mm^2, outperforming prior works by 1.2× to 16.9× in energy efficiency and 3.9× to 10.6× in area efficiency. Compared to our design, HyAtt achieves comparable macro-level energy efficiency using analog IMCs without sparsity. However, it only evaluates self-attention performance and thus cannot support end-to-end inference. PFCS exhibits 1.7× higher system-level energy efficiency, but it excludes the ADCs and other peripheral circuits from its evaluation. Overall, our work achieves superior accuracy, real-time latency, and competitive scalability, thereby paving the way for efficient large-scale LLM acceleration.

VII. CONCLUSION

This paper presents a scalable sparsity-aware transformer accelerator with an in-memory butterfly zero-skipper and a local-attention reusable engine. The butterfly-network-based feed-forward architecture, together with the routing decoding algorithm, enables irregularly sparse weight matrix multiplication while sustaining high compute utilization in IMCs. The local-attention reusable engine, combined with the *QK* sharing algorithm, enhances both data and computation reuse while eliminating redundant self-attention operations. The measurement results show that the fabricated accelerator in 28nm CMOS achieves 4.2× speedup and 27.6× energy efficiency over the Nvidia A100 GPU. Compared to state-of-the-arts, the proposed accelerator further achieves 1.2× to 16.9× energy efficiency.

REFERENCES

[1] J. Bai *et al.*, "Qwen technical report," 2023. [Online]. Available: https://arxiv.org/abs/2309.16609

[2] A. Q. Jiang *et al.*, "Mixtral of experts," 2024. [Online]. Available: https://arxiv.org/abs/2401.04088

[3] Y. Liu *et al.*, "Sora: A review on background, technology, limitations, and opportunities of large vision models," 2024. [Online]. Available: https://arxiv.org/abs/2402.17177

[4] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," 2020. [Online]. Available: https://arxiv.org/abs/2006.11239

[5] J. Jumper *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[6] J. Abramson *et al.*, "Accurate structure prediction of biomolecular interactions with alphafold 3," *Nature*, vol. 630, no. 8016, pp. 493–500, 2024.

[7] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[8] A. Josh *et al.*, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[9] A. Gholami *et al.*, "Ai and memory wall," *IEEE Micro*, vol. 44, no. 3, pp. 33–39, 2024.

[10] S. R. Agrawal *et al.*, "A many-core architecture for in-memory data processing," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 245–258.

[11] Y. Zhan *et al.*, "Gslp-cim: A 28-nm globally systolic and locally parallel cnn/transformer accelerator with scalable and reconfigurable edram compute-in-memory macro for flexible dataflow," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 72, no. 4, pp. 1657–1667, 2025.

[12] X. Yang *et al.*, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[13] J. Park *et al.*, "Attacc! unleashing the power of pim for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024, pp. 103–119.

[14] D. Kim *et al.*, "Dpim: A 19.36 tops/w 2t1c edram transformer-in-memory chip with sparsity-aware quantization and heterogeneous dense-sparse core," in *IEEE European Solid-State Electronics Research Conference (ESSERC)*, 2024, pp. 141–144.

[15] F. Tu *et al.*, "Multcim: Digital computing-in-memory-based multimodal transformer accelerator with attention-token-bit hybrid sparsity," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 1, pp. 90–101, 2023.

[16] A. Yazdanbakhsh *et al.*, "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 744–762.

[17] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[18] J. Yue *et al.*, "An energy-efficient computing-in-memory nn processor with set-associate blockwise sparsity and ping-pong weight update," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 5, pp. 1612–1627, 2024.

[19] C. Chu *et al.*, "Pim-prune: Fine-grain dcnn pruning for crossbar-based process-in-memory architecture," in *ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[20] J. Yue *et al.*, "Sticker-im: A 65 nm computing-in-memory nn processor using block-wise sparsity optimization and inter/intra-macro data reuse," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 8, pp. 2560–2573, 2022.

[21] J.-H. Kim *et al.*, "Z-pim: A sparsity-aware processing-in-memory architecture with fully variable weight bit-precision for energy-efficient deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 4, pp. 1093–1104, 2021.

[22] D. E. Kim *et al.*, "Samba: Sparsity aware in-memory computing based machine learning accelerator," *IEEE Transactions on Computers*, vol. 72, no. 9, pp. 2615–2627, 2023.

[23] W. Niu *et al.*, "Patdnn: Achieving real-time dnn execution on mobile devices with pattern-based weight pruning," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 907–922.

[24] N. Kitaev, Łukasz Kaiser, and A. Levskaya, "Reformer: The efficient transformer," 2020. [Online]. Available: https://arxiv.org/abs/2001.04451

[25] S. Liu *et al.*, "A 28nm 53.8tops/w 8b sparse transformer accelerator with in-memory butterfly zero skipper for unstructured-pruned nn and cim-based local-attention-reusable engine," in *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 250–252.

[26] L. Lu and otherss, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, pp. 977–991.

[27] H. Cho *et al.*, "Sparc: Token similarity-aware sparse attention transformer accelerator via row-wise clustering," in *ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[28] T. Dao *et al.*, "Flashattention: Fast and memory-efficient exact attention with io-awareness," 2022. [Online]. Available: https://arxiv.org/abs/2205.14135

[29] D. Guo *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[30] J. Ainslie *et al.*, "Gqa: Training generalized multi-query transformer models from multi-head checkpoints," 2023. [Online]. Available: https://arxiv.org/abs/2305.13245

[31] Y. Zhao *et al.*, "Atom: Low-bit quantization for efficient and accurate llm serving," 2024. [Online]. Available: https://arxiv.org/abs/2310.19102

[32] S. Cai *et al.*, "Adaptive two-range quantization and hardware co-design for large language model acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 15, no. 2, pp. 272–284, 2025.

[33] R. Hwang *et al.*, "Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference," in *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*, 2025, pp. 1018–1031.

[34] R. Sarkar *et al.*, "Edge-moe: Memory-efficient multi-task vision transformer architecture with task-level sparsity via mixture-of-experts," in *EEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 01–09.

[35] M. Sun *et al.*, "A simple and effective pruning approach for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2306.11695

[36] E. Frantar and D. Alistarh, "Sparsegpt: massive language models can be accurately pruned in one-shot," in *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023, pp. 10 323–10 337.

[37] E. Qin *et al.*, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 58–70.

[38] G. Li *et al.*, "Ristretto: An atomized processing architecture for sparsity-condensed stream flow in cnn," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 1434–1450.

[39] R. Hwang *et al.*, "Grow: A row-stationary sparse-dense gemm accelerator for memory-efficient graph convolutional neural networks," in *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023, pp. 42–55.

[40] R. Anil *et al.*, "Gemini: A family of highly capable multimodal models," 2025. [Online]. Available: https://arxiv.org/abs/2312.11805

[41] S. Agarwal *et al.*, "gpt-oss-120b and gpt-oss-20b model card," 2025. [Online]. Available: https://arxiv.org/abs/2508.10925

[42] F. Zhang *et al.*, "Efficient memory integration: Mram-sram hybrid accelerator for sparse on-device learning," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[43] Y. Ma *et al.*, "Hnm-cim: An algorithm-hardware co-designed sram-based cim for transformer acceleration exploiting hybrid n:m sparsity," *ACM Transaction on Design Automation of Electronic Systems*, vol. 30, no. 3, pp. 1–22, Apr. 2025.

[44] X. Kang *et al.*, "Suarch: Accelerating layer-wise n:m sparse pattern with a unified architecture for deep-learning edge device," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025, pp. 700–705.

[45] H. Zhu *et al.*, "Comb-mcm: Computing-on-memory-boundary nn processor with bipolar bitwise sparsity optimization for scalable multi-chiplet-module edge machine learning," in *IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[46] S. Sukhbaatar *et al.*, "Adaptive attention span in transformers," 2019. [Online]. Available: https://arxiv.org/abs/1905.07799

[47] G. Xiao *et al.*, "Smoothquant: Accurate and efficient post-training quantization for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2211.10438

[48] J. M. Springer *et al.*, "Overtrained language models are harder to fine-tune," 2025. [Online]. Available: https://arxiv.org/abs/2503.19206

[49] F. Tu *et al.*, "Trancim: Full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 6, pp. 1798–1809, 2023.

[50] G. Yin *et al.*, "Hybrid sram/rom compute-in-memory architecture for high task-level energy efficiency in transformer models with 8928-kb/mmtexpreserve0 density in 28nm cmos," *IEEE Journal of Solid-State Circuits*, pp. 1–14, 2025.

[51] A. Moradifirouzabadi, D. S. Dodla, and M. Kang, "An analog and digital hybrid attention accelerator for transformers with charge-based in-memory computing," in *IEEE European Solid-State Electronics Research Conference (ESSERC)*, 2024, pp. 353–356.

[52] Y. Qiu, G. Li, M. Wu, Y. Jia, L. Ye, and Y. Ma, "Quartet: A digital compute-in-memory versatile ai accelerator with heterogeneous tensor engines and off-chip-less dataflow," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2025.

[53] G. Singh and S. Vrudhula, "A scalable and energy-efficient processing-in-memory architecture for gen-ai," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 15, no. 2, pp. 285–298, 2025.