2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-2027-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/HPCA53966.2022.00082

TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer

Minxuan Zhou[§], Weihong Xu[§], Jaeyoung Kang, and Tajana Rosing
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA, United States
{*miz087, wexu, j5kang, tajana*}*@ucsd.edu*

***Abstract*—Transformer-based models are state-of-the-art for many machine learning (ML) tasks. Executing Transformer usually requires a long execution time due to the large memory footprint and the low data reuse rate, stressing the memory system while under-utilizing the computing resources. Memory-based processing technologies, including processing in-memory (PIM) and near-memory computing (NMC), are promising to accelerate Transformer since they provide high memory bandwidth utilization and extensive computation parallelism. However, the previous memory-based ML accelerators mainly target at optimizing dataflow and hardware for compute-intensive ML models (e.g., CNNs), which do not fit the memory-intensive characteristics of Transformer. In this work, we propose TransPIM, a memory-based acceleration for Transformer using software and hardware co-design. In the software-level, TransPIM adopts a token-based dataflow to avoid the expensive inter-layer data movements introduced by previous layer-based dataflow. In the hardware-level, TransPIM introduces lightweight modifications in the conventional high bandwidth memory (HBM) architecture to support PIM-NMC hybrid processing and efficient data communication for accelerating Transformer-based models. Our experiments show that TransPIM is 3.7× to 9.1× faster than existing memory-based acceleration. As compared to conventional accelerators, TransPIM is 22.1× to 114.9× faster than GPUs and provides 2.0× more throughput than existing ASIC-based accelerators.**

***Keywords*-Processing in-memory; Near-data processing; Transformer; Domain-specific acceleration; Software-hardware co-design.**

I. Introduction

Attention mechanism has emerged as a powerful tool to model long-term dependencies in sequential data [41]. Attention-based models, such as Transformer [41] and its enhanced variants [10], [36], have dramatically improved the accuracy of important machine learning tasks, like natural language processing [30], computer vision [6], [11], and video analysis [5]. However, these benefits come at the cost of long execution time due to the large memory footprint and low computation to memory ratio. Existing CNN-oriented accelerators are designed for compute-intensive operations (e.g., convolution), making them sub-optimal for processing Transformers [15], [42]. To accelerate Transformer models, there have been several domain-specific accelerators, such as SpAtten [42] and A^3 [15], that offload either the key operation (i.e., self-attention) or the whole Transformer from conventional systems (e.g., GPU). However, these ASIC-based accelerators still suffer from the constrained parallelism and limited off-chip memory bandwidth that bound the performance of acceleration.

Memory-based acceleration, including processing in-memory (PIM) and near-memory computing (NMC), is promising to accelerate Transformer models as it supports extensive parallelism, low data movement cost, and scalable memory bandwidth [1], [12], [14], [18], [29], [47], [49], [50]. Although there have been many memory-based neural network accelerators [12], [17], [27], [29], [37], their dataflow and hardware are mainly optimized for compute-intensive CNNs, which may be incompatible with memory-intensive Transformers. Specifically, the dataflows of existing memory-based accelerators [12], [17], [29] are determined in a layer-level granularity, such that either utilize the whole memory to process one layer at a time [12], [29] or allocate mutually exclusive memory resources to different layers [17]. Such layer-based dataflows introduce large non-compute overhead in Transformers because of the large amount of input data and weights that need to be loaded or transferred between layers. On the hardware side, Transformer's complex operations (such as reduction and Softmax), which require both parallel computation and fine-grained intra-memory data reorganization, would be the performance bottleneck for memory-based accelerators. Our experiments (Section II-C) show that the layer-based dataflows spend over 60% of execution time on data movements when accelerating a widely-used Transformer using an in-memory bit-serial accelerator on emerging high bandwidth memory (HBM). Furthermore, the reductions using bit-serial row-parallel PIM operations take around 30% of execution time, exhibiting a much lower compute throughput than other PIM arithmetic operations. The results show both software and hardware issues can significantly limit the efficiency of memory-based Transformer acceleration.

In this work, we propose TransPIM, a software-hardware co-design based on an emerging commodity memory, high

[§]Equal contribution

978-1-6654-2027-3/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCA53966.2022.00082

bandwidth memory (HBM), that utilizes memory-based acceleration technology to accelerate Transformers. In the software, instead of allocating memory for different layers, TransPIM adopts a token-based dataflow that assigns memory resources for computations across different layers based on the input tokens. With the token-based dataflow, each memory bank processes and stores the intermediate results related to a specific set of tokens. By doing so, TransPIM can significantly reduce the amount of data loaded or transferred across layers because the increased locality of intermediate data from different layers improves data reuse rate. TransPIM only requires data movement for computing the cross-memory (i.e., between different sets of tokens) information which can be handled efficiently by exploiting the large internal memory bandwidth of HBM.

Even though the token-based dataflow significantly improves the throughput by reducing the data movement overhead, the software-level solution cannot resolve the inefficiency of complex operations in Transformer. Existing memory-based accelerators supports either PIM [2] or NMC [16], [27]. However, different key operations in Transformers do not share similar patterns that can be efficiently processed by a single type of memory-based technology. Furthermore, the original data path in HBM heavily relies on the shared bus. Therefore, the resource conflict on the shared bus for transferring data may become the bottleneck for applications requiring high internal bandwidth. To solve these issues, we propose an integrated set of hardware in HBM, including near-memory auxiliary computing units (ACUs) and an optimized data communication architecture. Specifically, the ACUs enable the memory to exploit the benefits of both PIM and NMC to efficiently accelerate different operations. The optimized data communication architecture adds buffers and specialized links in the HBM hierarchy to offload a large number of data movements from the global shared data path, delivering significantly higher memory bandwidth utilization than the original HBM.

In summary, the contributions of this work include:

- Compared to existing accelerators [15], [42] dedicated to attention, TransPIM is the first end-to-end memory-based accelerator that speeds up the entire Transformer inference by exploiting the emerging memory-based acceleration.
- The proposed software-hardware co-design significantly outperforms existing platforms, including GPU, TPU, and ASIC-based accelerators. Specifically, TransPIM is 22.1 × to 114.9 × faster than GPUs on various widely-used Transformers. As compared to ASIC-based accelerators, TransPIM provides 2.0× higher throughput.
- We propose a token-based dataflow for general Transformer-based models to reduce unnecessary data loading by exploiting holistic data reuse. Our results show that the proposed dataflow is 4.6× faster than the previous method on various memory-based accelerator architectures.

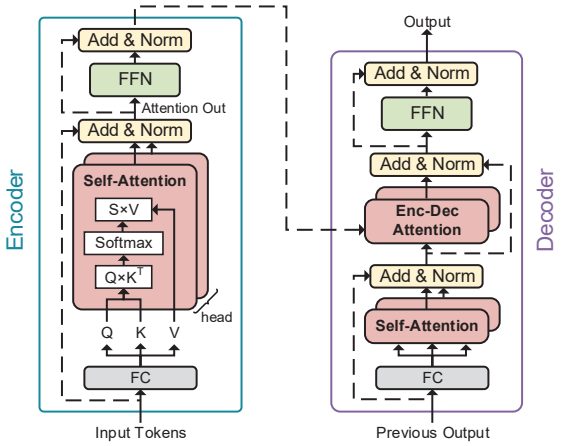


Figure 1. Operations of encoder and decoder blocks in Transformers.

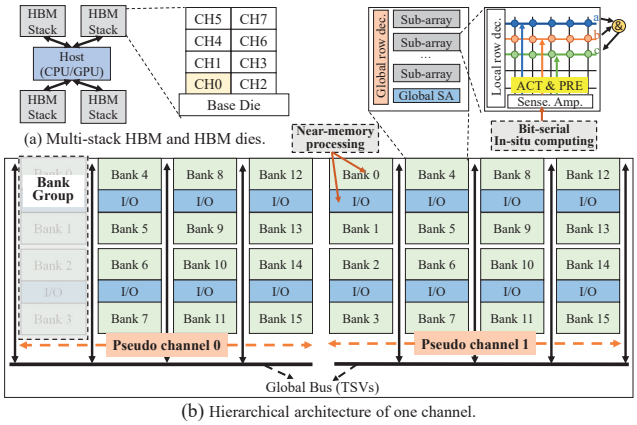
- TransPIM introduces lightweight hardware components in the conventional HBM architecture to support efficient computing and memory operations for Transformers, without impacting the memory density. Our experiments show that TransPIM significantly improves the performance by 3.7× and 9.1× over PIM-only and NMC-only architectures.

II. Background and Motivation

This section first introduces the background for Transformer and PIM acceleration. Then, this section analyzes the issues in existing memory-based technologies with accelerating Transformer-based models, which motivate a new software-hardware co-design.

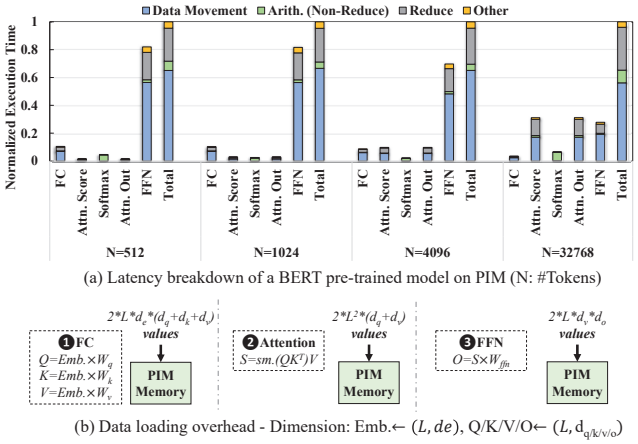
A. Transformer

Transformer has an encoder-decoder architecture [41], as shown in Figure 1. Both the encoder and the decoder are constructed by stacking identical blocks. Each encoder block has three sub-layers including a fully-connected layer (FC), a self-attention layer (SA), and a feed-forward layer (FFN). For an input sequence with L tokens, the FC layer gets an embedding matrix of dimension $L \times d_e$ and generates a query matrix Q of dimension $L \times d_q$ and a key matrix K of dimension $N \times d_k$, and a value matrix V of dimension $L \times d_v$ by multiplying the embedding matrix with different weight matrices. For simplicity, we use D to denote d_k, d_q and d_v. Each D-dimension vector in the Q, K and V matrices corresponds to an input token. The encoder block then feeds the Q, K, and V matrices to the SA layer. The key operation in a SA layer is the scaled dot-product attention which computes dependencies between input tokens as $\text{Softmax}(\frac{QK^T}{\sqrt{D}})V$, where $\text{Softmax}(\cdot)$ denotes the Softmax function. The $\frac{QK^T}{\sqrt{D}}$ is defined as the score matrix S. The encoder block feeds the attention output to the FFN layer to generate the block output, which can be used as an input to the next block (either encoder or decoder) or to a task-specific output layer (e.g., classification). Each decoder block also has the FC layer, the SA layer, and the FFN layer to process the output



(b) Hierarchical architecture of one channel.

Figure 2. Memory-based computing on HBM architectures.



(a) Latency breakdown of a BERT pre-trained model on PIM (N: #Tokens)

(b) Data loading overhead - Dimension: Emb.← (L, de), Q/K/V/O← $(L, d_{q/k/v/o})$

Figure 3. Challenges of PIM acceleration for Transformers.

of the preceding layer. Unlike encoder blocks, the input and output of decoder blocks usually contain only one or a few tokens. In addition, the decoder inserts another attention layer that performs attention over previous blocks. Transformer performs many memory-intensive operations [15], [20], [42], which make it suitable for in- and near-memory acceleration.

B. PIM Acceleration for Transformers

Memory-based computing has been widely used for various memory-intensive and compute-intensive applications due to its extensive parallelism and ability to minimize data movements. The baseline memory architecture used in this work is the Samsung's high-bandwidth memory (HBM) [27], [33] which has become the state-of-the-art memory solution for various emerging platforms [8], [23], [28], [51]. Figure 2 shows the overall architecture of 4 HBM stacks. All stacks are connected to the host CPU/GPU for cross-stack communication. Each HBM stack is a 4-high HBM chip with multiple DRAM slices on the top of the base die, connected with many through-silicon vias (TSVs), providing much higher bandwidth and lower access latency than the conventional DRAM.

There are two ways to process data in HBM. The first one is near-memory computing (NMC), where compute logic is integrated either in the near-bank I/O or, more aggressively, in the near-subarray circuits inside the memory bank. For example, Samsung recently proposed a new type of HBM called function in-memory DRAM (FIMDRAM) [27], that integrates programmable computing units (PCUs) in the I/O circuits of the memory banks. These non-trivial PCUs take up the chip area for some memory banks, decreasing the memory density. The second technology is processing in-memory (PIM) which supports computing in the DRAM banks (or subarrays) by specialized sequences of activate and pre-charge commands [14] or modified subarray structures [2], [29]. PIM usually requires data to be placed column-wisely and follows a bit-serial row-parallel scheme to process the computation. PIM provides a higher level of parallelism with fewer data movements than NMC, but can only support a limited set of operations with high latency (because of bit-serial processing). NMC supports more general operations but the throughput is limited by the number of NMC processing elements as well as the bandwidth of the data link.

C. Motivation of Software-hardware Co-Design

Many previous works show that both the software-level scheduling (a.k.a., dataflow) and the hardware design play important roles in neural network acceleration [26], [35], [43]. To maximize the parallelism, existing memory-based DNN accelerators [2], [12], [17], [29] adopt a layer-based dataflow which allocates sufficient memory resources to parallelize computations for different output elements in a layer. The layer-based dataflow requires a whole data loading before processing each layer. We conduct an investigation on the efficiency of existing memory-based acceleration for Transformers. We evaluate the latency breakdown of a text classification task using RoBERTa Transformer model on an HBM-based PIM-only system which has 8 8GB HBM stacks. The PIM-only system processes all computations using bit-serial row-parallel operations inside memory subarrays. Figure 3(a) shows the profiling results. Figure 3(b) shows the size of loaded data for each layer in Transformer when using layer-based dataflow.

Our experiments show that the data movement of the layer-based dataflow takes the majority (around 60%) of the execution time, which is the time for loading and reorganizing data. The long data movement time results from two aspects. First, to maximize the parallelism, the layer-based dataflow needs data duplication in the memory for parallel computations, increasing the amount of loaded data. Second, most parallel data layouts do not exploit the data reuse between neighboring layers. In this case, we need to load all data for the intermediate layers (e.g., the attention layer). As shown in Figure 3(b), the size of computation data for the attention layer grows quadratically with the sequence length. Therefore, minimizing the amount of loaded data is critical to reducing the overall execution time of Transformer.

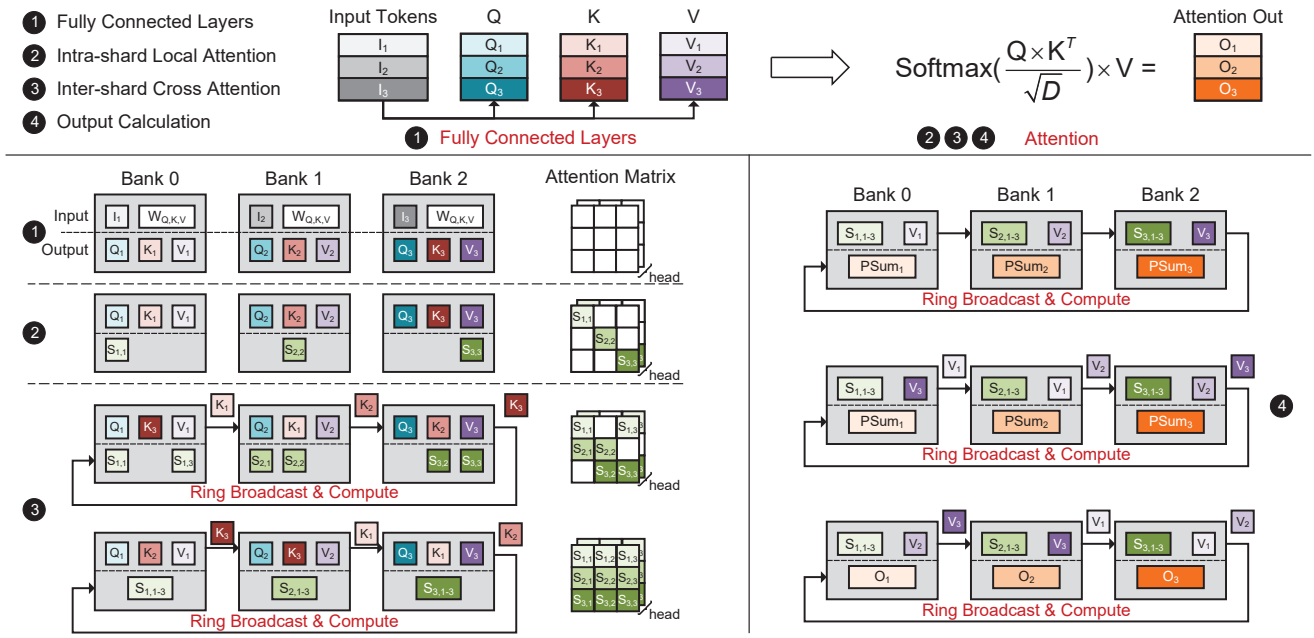


Figure 4. Token-based data sharding scheme and the dataflow of Transformer encoder in TransPIM. Banks = 3.

In this work, we exploit the fact that all operations in different Transformer layers are related to tokens in the input sequence, making it possible to improve the data locality by reusing token-related data during the execution.

In addition to the data loading issue, the evaluated PIM-only accelerator does not perform well for some complex Transformer computation primitives, like reduction operations, due to the costly intra-bank (or subarray) data movements for long vectors in Transformer (e.g., $D = 512$). For example, the reduction takes 23% to 32% of time for different sequence lengths, which is significantly larger than other arithmetic operations (4% to 10%). This is because the reduction requires data movements to reorganize the data in the memory for accumulation, degrading the efficiency of PIM operations. Such intra-array data movements introduce more overhead in Transformers than CNNs because the length of vectors for reduction is much longer in Transformers (e.g., 512 for Transformer vs. 9 for convolution with 3×3 kernels).

D. Key Ideas of TransPIM

Our investigation shows that existing technologies introduce large overhead on Transformers, requiring specific modifications on both software dataflow and hardware support. Therefore, this work proposes to accelerate Transformer via a software-hardware co-design.

Dataflow: TransPIM adopts an efficient dataflow which maps Transformer computation to the memory-based architecture using a token-based sharding mechanism. TransPIM divides the input tokens into different shards and allocates these shards to different memory partitions. In this work, we use memory bank as the basic memory partition for shard allocation. During acceleration, each memory partition processes its associated token shard independently across different layers. As compared to layer-based dataflow, the token-based dataflow avoids the memory traffic for reused data. We also propose an efficient broadcasting algorithm to speed up data movements for dependent data by exploiting the large internal memory bandwidth of HBM.

Hardware acceleration: In the hardware, TransPIM adds lightweight modifications to the conventional HBM architecture, which not only accelerate various Transformer operations but also efficiently support the proposed dataflow. Specifically, TransPIM architecture implements multiple auxiliary computing units (ACUs) within each memory bank to perform vector reduction and Softmax function that cannot be efficiently processed by bit-serial row-parallel PIM operations. TransPIM exploits the benefits of both PIM and NMC to achieve high efficiency and throughput. Furthermore, TransPIM enhances the original HBM data path with specialized data buffers and communication links to support various data manipulations and movements for Transformers.

III. TransPIM Dataflow

In this section, we introduce the detailed process of TransPIM dataflow. The underlying architecture is based on compute-enabled HBM as shown in Figure 2.

A. Token-based Data Sharding

The key of TransPIM dataflow is token-based data sharding, which allocates HBM banks based on input tokens. The main benefits provided by the token-based data sharding come from

the data reuse across different layers by keeping computations of tokens in the same memory location. We can reduce the data movement cost while exploiting more memory-level parallelism because different banks can handle computations and data movements for allocated tokens independently. After the token sharding, each bank handles computations for its shards throughout the end-to-end Transformer inference. The token-based data sharding is applied to the input tokens before the fully-connected layers of the first encoder block. As introduced in Section II, the input tokens form a matrix with size $L \times D$, where L is the number of tokens and D is the embedding vector dimension. The input tokens are uniformly divided into "shards" along the token dimension and allocated to different memory banks. In the case of N memory banks, $\frac{L}{N}$ tokens are assigned to each bank. Therefore, each bank receives an input matrix with size $\frac{L}{N} \times D$. Figure 4 shows an example of distributing 3 tokens into 3 memory banks.

B. Encoder Blocks

During the Transformer inference, each memory bank performs the computations of FC, attention, and FFN layers for its allocated tokens.

1) Fully-Connected Layer: The FC layer generates the query (Q), key (K), and value (V) matrices from the input tokens. It involves three matrix multiplications between the input tokens and three weight matrices (W_Q, W_Q, and W_Q). In TransPIM, all weights of the three FC weight matrices are loaded into each memory bank before FC computation. And each bank multiplies the assigned $\frac{L}{N} \times D$ sub-matrix I_i with $D \times D$ weight matrices as illustrated in (1) of Figure 4. Then, each bank generates three $\frac{L}{N} \times D$ sub-matrices, Q_i, K_i, and V_i. The Q_i, K_i, and V_i matrices are retained in each memory bank and used for the following attention layers.

2) Attention Layer: The computation of attention layer in TransPIM involves three steps: (1) *Intra-shard local attention*, (2) *Inter-shard cross attention*, and (3) *Softmax*.

Intra-shard local attention: The attention scores S are computed by $S = Q \times K^T$. In TransPIM, Q and K matrices are distributed in memory banks. With the local sub-matrices, each memory bank first computes the attention scores between local tokens, as shown in (2) of Figure 4). Each memory bank computes the $\frac{L}{N} \times \frac{L}{N}$ partial attention scores in the diagonal of attention score matrix using the local Q_i and K_i from the FC layer. During the intra-shard local attention, each bank i can independently compute the partial attention score matrix $S_{i,i}$ without communicating with other banks.

Inter-shard cross attention: After computing all local attention scores, TransPIM computes other attention scores by moving partial K matrices between different banks. As shown in (3) of Figure 4, the inter-shard cross attention consists of multiple ring broadcast and compute steps. To improve the bandwidth utilization, we propose a ring broadcast scheme to transfer K_j data between banks, where each K_j sub-matrix is sent to all banks step by step through an abstract ring of banks (e.g., 0→1→2 ... N→0). In each broadcast step, each bank multiplies local Q_i with the received K_j from a remote bank, generating an $\frac{L}{N} \times \frac{L}{N}$ partial attention scores in the i-th row and j-column of the blocked attention matrix. A total of N ring broadcast and compute steps are required to obtain the entire attention score matrix S. Each bank preserves $\frac{L}{N}$ rows of the attention score matrix, S_i, with shape $\frac{L}{N} \times L$. The performance of inter-shard cross attention heavily depends on the speed of the ring broadcast phase. In Section IV-B, we provide an efficient hardware design and a scheduling scheme to fully exploits the internal memory bandwidth of HBM for ring broadcast-based data transfer.

Softmax: The Softmax layer normalizes the exponential of attention scores related to each token. Each bank calculates the Softmax using its local $\frac{L}{N}$ rows of attention scores. There is no data movement between banks thanks to the data locality of attention scores. For multi-head attention with h heads, there are h attention matrices to calculate. Therefore, the Softmax should be repeated h times to obtain all the results. The naive Softmax requires complex exponential function, reduction, and division. Thus, we design an efficient approach in Section IV-A2 for calculating Softmax function in the TransPIM hardware.

3) Self-attention Output and Feed-forward Network: The final step of the self-attention is to multiply the attention score matrix S after Softmax by the V matrix. With the token-based data sharding, each bank stores the partial attention scores and partial V_i matrix. The calculation of self-attention output ((4) of Figure 4) is similar to inter-shard cross attention. The partial V_i matrix is broadcasted through banks and each bank computes the $\frac{L}{N} \times D$ partial attention out O_i. Feed-forward network (FFN) consists of two consecutive FC layers. In the last step of Figure 4(4), the attention out (input of FFN) has the same token sharding as input FC layers ((1)). Therefore, each FFN-FC layer has a similar process to the input FC layers.

C. Decoder Blocks

The key difference between the decoder block and the encoder block is that each decoder block only needs to calculate one new token and attention operations between the new token and the old tokens. Figure 5 shows the processing flow for decoder layers which is slightly different from the encoder blocks. In each decoder block, we keep using the data sharding of the preceding block, which can be either the last encoder or the previous decoder. We allocate the last bank to process the FC layers for the new Q, K, and V vectors. In (1) of Figure 5, we allocate Bank 2 to process Q_{new}, K_{new}, and V_{new}. At the end of FC layers, TransPIM sends the generated Q_{new} to all other banks to compute attention scores. Besides, K_{new} and V_{new} are concatenated to the old K_i and V_i of last bank. At (2), each bank performs intra-shard local attention using local Q_{new}, K_i, and V_i. At the end of (2), each block obtains and preserves $\frac{L}{N}$ columns ($\frac{L}{N}+1$ for the last bank) of

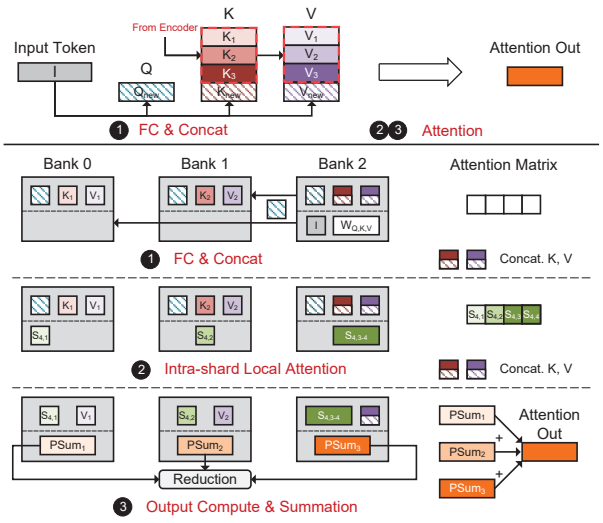


Figure 5. Dataflow of TransPIM for Transformer decoder.

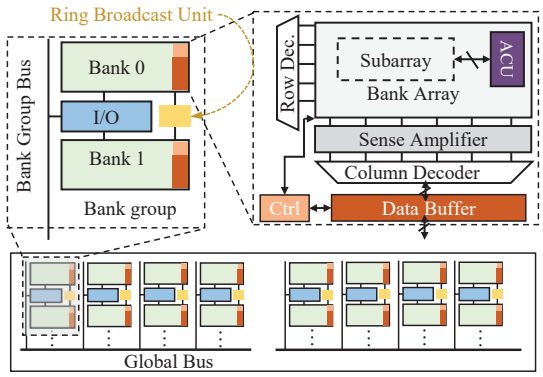


Figure 6. Overview of TransPIM hardware based on HBM.

the new score matrix. In the output compute and summation (3), each bank uses its local S_i and V_i vectors to compute partial sum $PSum_i$ of the new attention output O_{new}. This scheme requires one reduction step to generate the correct output. In Section IV-B, we introduce how to efficiently process such reduction in a modified HBM architecture. The decoder scheme supports both encoder-decoder and decoder-only Transformers. The only difference between these two types is whether memory banks pre-store "context" vectors which are K and V vectors from encoder blocks. For each new token, we allocate the bank with the minimum number of tokens to balance computation.

IV. TransPIM Hardware Acceleration

We propose a new memory-based hardware acceleration to address the challenges of accelerating Transformer models. Figure 6 shows the hardware customization in the standard HBM2 structure [19], which has two parts – 1) in-bank auxiliary computing units (ACUs), 2) a data communication architecture with near-bank data buffer and ring broadcast units integrated into the original HBM data path.

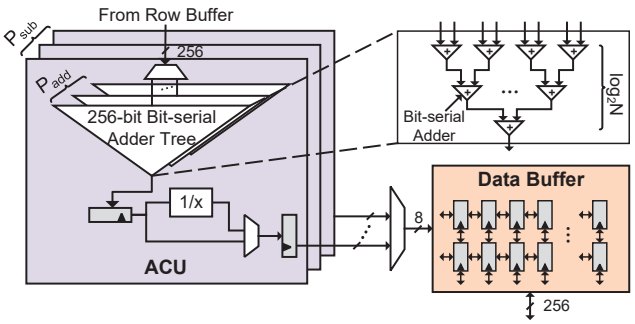


Figure 7. Detailed design of ACU and data buffer.

A. Auxiliary Computing Unit

TransPIM adds ACUs to support operations that are not friendly for in-memory bit-serial operations.

1) ACU Design: Unlike previous near-bank processing units [16], [27] for conventional row-wise data layout, the proposed ACU works on column-wise bit-serial data in order to support in-memory operations. As compared to previous PIM-only work [29], which modifies the cell array structure and adds extra shifters, the ACU offloads reduction, Softmax, and data broadcast with a more light-weight design.

As shown in Figure 7, each ACU is concatenated after the subarray to receive the 256-bit data from the row buffer because it is too expensive to employ arithmetic units to simultaneously process the data of the entire row buffer (e.g., 8Kb). Therefore, the same bit of 256 different numbers in a row is accessed from the row buffer for each memory access to support the bit-serial data layout for in-memory operations. A total of P_{add} 256-bit adder trees are implemented within each ACU to fully exploit the internal bandwidth of memory bank and reduce the DRAM's row activation overhead. Each adder tree is composed of area-efficient 255 bit-serial adders [3] to support the bit-serial data organization and reduce overhead. Besides, the bit-serial adder tree is stage-pipelined. A register and a divisor are implemented to latch intermediate partial sums and compute the reciprocal of row accumulation in Softmax function. Similar to [29], P_{sub} subarrays in each bank are activated simultaneously to increase the computation parallelism. Thus P_{sub} ACUs are implemented for each memory bank.

When the vector length of point-wise vector multiplication results is less than the width of the adder tree, the ACU reduction can be computed using a single bit-serial adder tree. For b-bit data, a total of b row accesses are required to compute the reduction results. However, the reduced vector length is generally larger than the width of the adder tree. In this case, ACU needs to issue $b \times \lceil \frac{N}{256} \rceil$ row accesses, where N is the vector length (> 256). To speed up the ACU reduction and save energy dissipation, we implement P_{add} bit-serial adder trees in parallel within each ACU. Before precharging and activating a new row, ACU performs P_{add}-

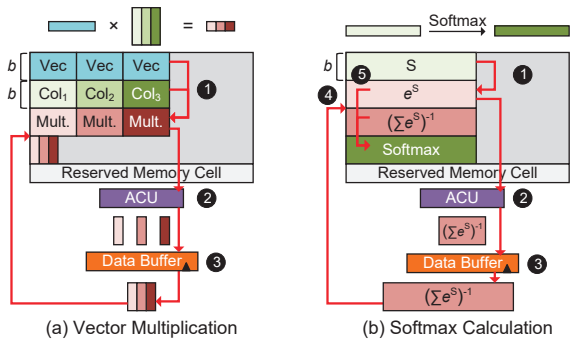


Figure 8. The data path of different computations in TransPIM.

time column accesses in the same row and consecutively feeds the data into P_{add} adder trees. Considering that the interval t_{CCD} for DRAM to issue column access commands is much less than row access t_{RC} (i.e. 20× less), the row activation time and latency of reduction decrease to around $\frac{1}{P_{add}}$ compared to the case with just one adder tree. Moreover, the proposed design trades excessive row activation energy by the register energy. The energy consumed by reduction operation is significantly reduced.

2) TransPIM Computing: TransPIM supports all computing operations for Transformer by adopting a hybrid in-memory and near-memory computing paradigm with ACUs and data buffers. Specifically, point-wise vector operations are performed in the memory cells since DRAM natively supports such operations with very high parallelism [14], [39]. TransPIM offloads the vector reduction and part of Softmax function from subarray to ACU to improve the efficiency.

Vector Computation: Figure 8 (a) shows the data path of vector multiplication in TransPIM. The vectors are organized in a bit-serial format, where b rows of the same bit line are allocated for each b-bit data. For vector multiplication, it includes two steps: point-wise multiplication and reduction. TransPIM calculates the point-wise multiplication in the memory array using existing schemes [2], [14], [39]. We adopt the Boolean majority functions [2] to reduce the computing latency. These PIM operations utilize DRAM timing violations to perform bulk bit-wise operations in the standard DRAM architecture [19]. The vector has three copies in b rows to parallelize the computation (1). The point-wise multiplications between the replicated vectors and columns of the matrix are computed simultaneously. After point-wise multiplication, a vector reduction is required to obtain the results. The concatenated ACU to the subarray calculates the reduction of point-wise multiplication results (2). The ACU continuously receives data in bit-serial format from the row buffer and temporarily store the reduction results in the data buffer. The final reduction results will be written back to the memory cell as 3.

Softmax Calculation: PIM is unable to directly support the complicated exponential and division operations of Softmax. We resolve this limitation in TransPIM by rewriting Softmax function as $\frac{1}{\sum_{l=1}^{N} e^{S_{i,j}}} e^{S_{i,j}}$, where the reciprocal of row accumulation is moved out of the point-wise exponent. In this case, Softmax becomes the multiplication between the point-wise exponent and the reciprocal of the associated row. The point-wise exponent of attention score matrix S should be first calculated. Then the final Softmax output is the point-wise division between the exponents and the accumulation in the associated row. As shown in 1 of Figure 8 (b), the point-wise exponent is approximated using five-order Taylor series expansion, which is computed by PIM multiplication and addition. Then row accumulation is offloaded to ACU using vector reduction. Meanwhile, the divider in ACU computes the reciprocal of row reduction (2). The single reciprocal value for a row will be replicated 256 copies and written back the memory cell array by the data buffer (3 and 4). Finally, the point-wise multiplication between reciprocal and exponent is computed in memory by PIM operations (5).

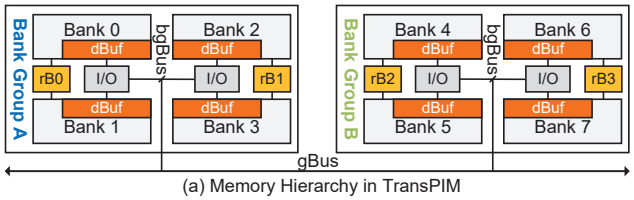
B. Data Communication Architecture

TransPIM dataflow exploits the internal memory bandwidth to reduce the data movement overhead caused by data loading. However, the standard HBM is still insufficient to match the high data parallelism and the internal bandwidth requirement. Even though we can use the bulk in-memory data movement approach like RowClone [38], the internal bandwidth is still limited by the shared data bus. Furthermore, the memory system needs frequent intra-bank and inter-bank data movements for memory-based computations, where data copy and re-organization may significantly downgrade the performance. Thus, we propose a data communication architecture to accelerate various data movements in TransPIM.

1) Customized Hardware Components: TransPIM introduce two customized hardware components in the HBM architecture for data communication.

Data Buffer: For most intra-bank and inter-bank data movements, we can use the fast parallel mode (FPM) of RowClone [38] to perform fast row copy. However, this approach has two defects. First, it is unable to provide a fine-grained partial copy for a row. Second, the FPM requires the source and destination rows to be located within the same subarray. To overcome the two constraints, we implement a re-configurable data buffer in each bank to manipulate data from ACU or row buffer as in Figure 7, realizing more flexible data movement. The data buffer is a configurable 8×256b buffer, consisting of 8 256-bit shift registers, supporting data copy and re-organization. The data buffer can either receive 8-bit (from ACU) or 256-bit data (from sense amplifier).

Ring Broadcast Unit: As illustrated in Section III, TransPIM adopts a ring-based data broadcast to reduce the data loading overhead for processing attention scores and self-attention output. However, the original HBM cannot efficiently support such ring-based data broadcast because all data transfers in a channel need to use the shared data bus and controller. TransPIM effectively decouples data transfers



(a) Memory Hierarchy in TransPIM

Time Step	1	2	3
Bank Group Bus A	3→4	7→0	1→2
Bank Group Bus B	3→4	7→0	5→6
Ring Broadcast Buffer	0→1, 6→7	2→3, 4→5	*idle*

(b) Timeline for Ring-based Broadcast

Figure 9. The optimizations for ring-based broadcast in the TransPIM hardware. The example shows the data transfers between two bank groups (4 banks per bank group).

between different bank groups (or even different banks) by per-bank broadcast units and direct 256-bit link between broadcast units of two neighboring banks, as shown in Figure 9(a).

2) Optimization for TransPIM Data Movement: With the proposed communication architecture, we can accelerate various data movement patterns when running TransPIM dataflow.

Fine-grained data movement: Each data buffer, with the help of its controller, supports fine-grained data copy and duplication in a bank. When fine-grained partial copy is needed (such as ❸ and ❹ in Figure 8 (b)), the data buffer reads data from ACU through 8-bit input and performs data replication. The replicated data is written back to the sense amplifier through 256-bit output in a bit-serial manner. Another advantage of data buffer is its ability to move data between different subarrays without using the shared bus. The data buffer supports parallel accesses by reading 256-bit data from the sense amplifier for each column access cycle. It can cache at most 2Kb data and copy each 256-bit data into the sense amplifier located in a different subarray.

Ring-based data broadcast: Figure 9 shows the data movements of ring-based data broadcast (Section III-B2) in two bank groups of the TransPIM architecture. As illustrated in Section III, each step of the ring-based broadcast requires all banks to copy data to their next banks in the ring (e.g., 1→2→3 ... 7→0 in the figure). If we assume the time of a data copy between two banks is T, the original HBM architecture requires $8T$ because each data copy requires the global bus and controller. For TransPIM architecture, such ring-based broadcast consumes a time of $3T$ as shown in Figure 9. In the first step, we use the bank group bus (both BankGroup A and BankGroup B) to perform bank 3→4 transfer. At the same time, we can also copy data from bank 0→1 and 6→7 using ring broadcast links between broadcast buffers. In the second step, we use the bank group bus to transfer 7→0, while using the ring broadcast buffers for 2→3 and 4→5. The two remaining transfers, 1→2 and 5→6 can be processed in parallel during the third step. The algorithm can scale to more bank groups with the same time complexity, which is significantly lower than that of the non-optimized architecture.

Table I
ARCHITECTURAL PARAMETERS FOR TRANSPIM

HBM Organization	Channels/die = 8, Banks/channel = 32, Banks/Group = 4, Rows = 32k, Row Size = 1KB, Subarray size = 512×512, DQ size = 256
HBM Timing (ns)	$t_{RC} = 45$, $t_{RCD} = 16$, $t_{RAS} = 29$, $t_{CL} = 16$, $t_{RRD} = 2$, $t_{WR} = 16$, $t_{CCD_S} = 2$, $t_{CCD_L} = 4$
HBM Energy (pJ)	$e_{ACT} = 909$, $e_{Pre\text{-}GSA} = 1.51$, $e_{Post\text{-}GSA} = 1.17$, $e_{I/O} = 0.80$
ACU	Clock = 500 MHz, $P_{sub} = 16$ ACUs/bank, $P_{add} = 4$ Pipelined Bit-serial Adder Tree/ACU, Adder tree width = 256, 3-stage pipelined divider
Buffer	Data buffer: 8×256b, Ring broadcast width = 256

Token reduction in decoder blocks: As introduced in Section III-C, the output token of each decoder block requires a global reduction for all partial sums distributed in different banks. TransPIM can efficiently reduce all the partial sums in a multi-step parallel way. Specifically, in each reduction step, we separate banks with partial sums into multiple two-bank reduction groups and reduce partial sums of each reduction group by moving partial sums from one bank to another. All reduction groups process the reduction in parallel with PIM operations. TransPIM can efficiently support such data movements by exploiting the internal bandwidth provided by inter-ACU links, bank group bus, and channel bus.

V. EXPERIMENTS

In this section, we describe our experiments that evaluate the benefits of proposed design.

A. Evaluation Methodology

The hardware characteristics for TransPIM are summarized in Table I. The memory is standard HBM2 [19]. The timing and energy parameters are extracted from the previously published work [34]. Hardware components of TransPIM keep the same area and power constraints as the original HBM. The HBM area is estimated using the analytical tool CACTI-3DD [7] on 22nm technology node. We assume up to 8 HBM stacks are connected to a host CPU through the silicon interposer. The host-HBM bandwidth is 256GB/s [34].

We implement TransPIM using Verilog HDL and synthesize the design on Synopsys Design Compiler using 65nm library. The synthesized design is placed and routed using Synopsys IC Compiler. Moreover, clock gating is applied to save energy dissipation. $P_{add} = 4$ bit-serial adder trees are implemented in each ACU. The constant divider to calculate $1/x$ is three-stage pipelined to satisfy the timing constraints. In order to match the rate of column access time $t_{CCD} = 2$ ns, the ACU is configurated to run at 500 MHz clock frequency. The obtained area and power data of ACU are scaled to 22nm to match the memory technology. We consider the process difference between logic and DRAM using the similar

Table II
OVERHEAD BREAKDOWN OF TRANSPIM.

Unit/Bank	Area (um^2)	Power (mW)
Adder Tree	59432.1	25.1
Divider	3055.6	0.7
Data Buffer	2660.4	3.8
Ring Broadcast	337.9	0.2
Others	828.5	2.9

TransPIM	Area (mm^2)
8GB HBM2	53.15
Overhead	2.15
Memory Access	**Energy (pJ/op)**
ACU	0.384
Buffers	0.869

method in previous work [29], where DRAM process incurs around 50% additional area overhead to the logic process.

The implementation results of TransPIM are given in Table II. The 4-parallel bit-serial adder tree takes up 88% of the overall area. Each memory bank of TransPIM is equipped with $P_{\text{sub}} = 16$ ACUs. The total 512 ACUs consume about 2.15 mm^2, incurring 4.0% area overhead to the original DRAM architecture, far less than the 25% threshold of area overhead [16], hence avoiding DRAM density loss.

1) Simulation: We implement an in-house simulator to model the detailed performance and energy characteristics for TransPIM and all PIM baselines. The front-end of the simulator utilizes the TensorFlow interface which extracts the workload formation for the simulation. The backend simulator is a modified version of Ramulator [24]. We insert additional commands to the simulator for TransPIM to simulate the run-time behaviors of workloads for a given DRAM configuration. The architectural configuration of HBM and timing/energy parameters are shown in Table I.

2) Hardware Baselines: **GPU&TPU:** The GPU platform is Nvidia RTX 2080Ti. We measure the GPU power using `nvidia-smi`. We also include a single Google Cloud TPUv3 with eight cores [22] as a baseline. We used JIT-compiled TensorFlow models and calculated the average latency from the second iteration to neglect graph compilation overhead.

Near-bank processing (NBP): Newton [16] is used as the near-memory baseline which is a near-bank processing technology in HBM2E-like DRAM offloading most operations for machine learning model to the near-bank logic. Since the NBP baseline already modifies the bank-level logic, we enable the broadcast buffer, which handles intra-memory data movements, in the NBP baseline for a fair comparison. We assume the same HBM architecture for the NBP baseline as the one used by TransPIM.

Original PIM: The original PIM architecture is the basic HBM architecture with only the support for in-memory bit-serial operations using the specialized memory controller with modifications to the subarray as suggested by previous works [2]. We also assume the same HBM architecture for the PIM baseline as TransPIM.

3) Workloads: In this work, we evaluate two widely used Transformer models, RoBERTa [31] and Pegasus [46], for various important NLP tasks including text classification (IMDB) [32], summarization (Pubmed [9] and Arxiv [9]), and question-answering (TraviaQA [21]). The classification and question-answering tasks only have encoder blocks while the summarization tasks have both encoder and decoder blocks. We also evaluate a decoder-only task, language modeling (LM), using GPT-2-medium model [36]. All workloads are implemented using TensorFlow 2 with XLA.

B. TransPIM Performance

We evaluate the efficiency of TransPIM by comparing it with GPU and various memory-based architectures with either layer-based or token-based dataflow. We denote each system as "dataflow"-"architecture" (e.g., `Token-TransPIM`). For sensitive analysis, we test one more architecture configuration of TransPIM which disables broadcast units and data buffers for communication – denoted by "NB", while "Buf" denotes architectures with broadcast units and data buffers. Figure 10 shows the performance and the energy efficiency of all architectures as compared to the GPU baseline. All memory-based systems use 8 HBM stacks with a total capacity of 64GB. The performance is measured as the execution time per batch because workloads with short token lengths (e.g., IMDB and TriviaQA) may not fully utilize the memory for just a single batch. The GPU system runs with the maximum batch size supported for each workload. The energy efficiency is measured as GOP/J of different systems. All values are normalized to the GPU baseline. All baselines run with a precision of 8-bit for FC and FFN layers which is sufficient for Transformer models [44]. We use 16-bit for Softmax to support a range of exponential.

Comparison to GPU/TPU: The proposed system (`Token-TransPIM`) is 22.1× (8.7×) to 114.9× (57.4×) faster than GPU (TPU). TransPIM shows less significant performance improvement on IMDB because the number of tokens is too small for the PIM system to fully exploit the parallelism because the token-based sharding requires each bank to process at least one token. For the workloads with more tokens, the token-based scheduling can saturate the compute capability of PIM system, fully exploiting the parallelism of PIM operations. As for the energy efficiency, TransPIM is 138.1× (39.5×) to 666.6× (376.7×) more energy efficiency than the GPU (TPU). Similar to the performance results, TransPIM achieves much better efficiency when running workload with long token sequence. The energy efficiency improvements result from the fast execution and the reduction of data movements.

Comparison to previous memory-based acceleration: As compared to previous PIM-only acceleration (layer-allocation), TransPIM with the token-sharding is 9.6× faster. If the PIM-only acceleration also uses the token-sharding processing, TransPIM is still 3.7 × faster. Furthermore, TransPIM is 4.2 × and 1.3× more energy efficient than the PIM-only acceleration with layer-based dataflow and token-sharding respectively. Such results show that TransPIM improves the performance and the energy efficiency of previous PIM acceleration by both software-side and hardware-side customization.

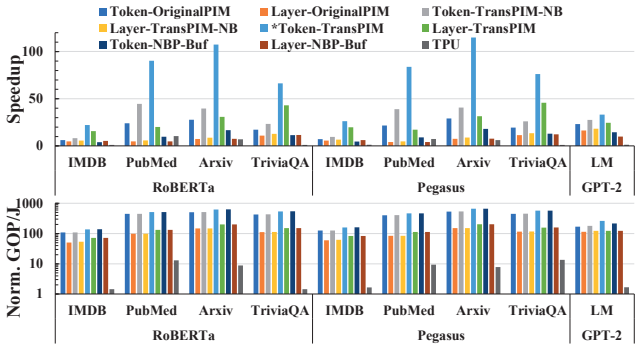


Figure 10. Performance and energy efficiency result.

As compared to the NBP architecture, TransPIM is 9.1× and 6.4× faster with token-sharding and layer-based dataflow respectively. However, TransPIM is not more energy-efficient than the NBP baseline with the same dataflow (around 0.2% less). This is due to the large amount of energy consumed by bit-serial in-situ operations which require a lot of parallel row activation and pre-charge operations for all memory subarrays.

Comparison to ASIC: The previous ASIC designs, A^3 [15] and SpAtten [42], adopt pruning techniques to reduce the computation complexity and mostly focus on accelerating the self-attention layers. TransPIM targets lower data movement overhead and higher computing efficiency for the end-to-end execution of Transformer models. Since previous ASIC counterparts neglect the area of memory, we assume all systems use the 8GB HBM as the memory. The additional area of TransPIM is $2.15mm^2$ for each 8GB HBM chip, which is close to A^3 ($2.08mm^2$) and $SpAtten_{1/8}$ ($1.55mm^2$). SpAtten [42] reports a 35× end-to-end performance improvement for generative stage (decoder) in GPT-2 model as compared to GPU. As a comparison, TransPIM achieves 83.9× and 114.9× speedup on two similar workloads (PubMed and Arxiv with Pegasus). Furthermore, TransPIM yields an average throughput of 734 GOP/s which is around $2.0-3.3\times$ of the peak throughput of A^3 (221 GOP/s) and SpAtten (360 GOP/s). The gain comes from three aspects. The token-based data sharding avoids redundant data movement, thus improving the computation efficiency. Moreover, the high data parallelism of in-memory and near-memory computing provides higher peak performance. The optimized data path of TransPIM exploits the large internal bandwidth of HBM to reduce the data movement overhead. In comparison, the performance of ASIC is constrained by limited computing resources and off-chip memory bandwidth.

Decoder-only model: For decoder-only workload (GPT2-LM), TransPIM is 1.4× faster and 2.1× more energy-efficient than the second-best system (`Layer-TransPIM`). Both speedup and energy efficiency improvement of TransPIM over other systems become less than other workloads. This results from the fact that the decoder-only model only processes 1 token in each iteration, requiring much less data loading for in-memory computations than encoder-based models.

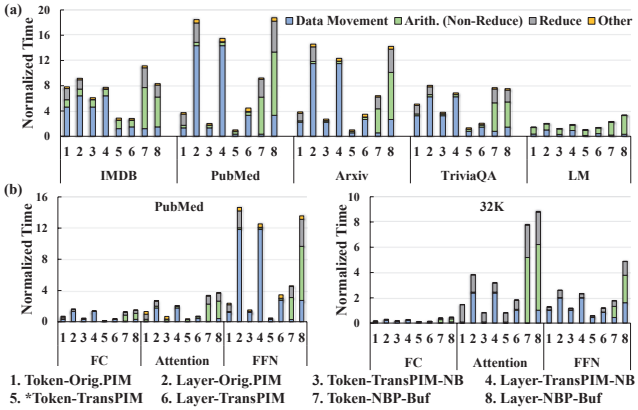


Figure 11. Performance breakdown of different systems: (a) overall breakdown, and (b) layer-wise breakdown.

C. Detailed Performance Analysis

We also investigate the detailed breakdown of operations for all memory-based systems, as shown in Figure 11. The figure shows the breakdown of four important categories of operations including the data movement (loading and intra-memory copy), non-reduction arithmetics, reduction, and other operations including reads and stores.

Improvements over previous acceleration: As compared to the PIM-only system, TransPIM significantly reduces the overhead of data movement because of the efficient data path (18.2× and 4.1× improvements for layer-based or token-sharding dataflow). Furthermore, the customized ACU of TransPIM effectively accelerates the costly reduction operations, where TransPIM spends 35.3× and 56.1× less time on reduction than PIM- and NBP-only systems. As compared to the NBP baseline, the performance improvement of reduction operation becomes even larger because the NBP baseline has a much lower degree of parallelism. The limited parallelism of the NBP baseline significantly increases the latency of other arithmetic operations as compared to the PIM implementation which is 13.2× faster.

Effect of token-sharding: The breakdown also sheds light on the performance benefits of token-sharding. For all systems, adopting token-sharding reduces the data movement latency by 4.8×, 4.5×, and 4.5× respectively. Such improvements depend on the workloads, where we observe 1.3×, 10.1×, 5.0× and 1.9× improvement on IMDB, PubMed, Arxiv, and TriviaQA respectively. Such results show that the token-sharding works better in large workloads (longer sequence) than small workloads because the data loading time of layer-allocation schemes increases quadratically with the sequence length for the attention layers. For the token-sharding, the size of moved data only increases linearly.

Effect of data movement optimization: While the token-sharding dataflow can significantly reduce the data

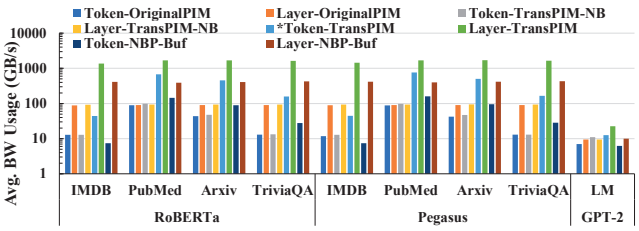


Figure 12. Average bandwidth usage.

movement overhead, TransPIM can further reduce it through the customized data path with broadcast and copy buffers. As compared to TransPIM without buffers, such customized data path provides a 4.1× reduction on the data movement.

Layer-wise breakdown: Figure 11(b) shows the layer-wise breakdown results of summarization using Peagasus for two workloads – PubMed (4K) and a synthetic data with 32K sequence length. All results are normalized to the total time of the proposed `Token-TransPIM` system. Token-based data sharding reduces the data movement overhead in FC and FFN layers because it requires less data duplication for computation (but less parallelism) than layer-allocation dataflow. In attention layers, TransPIM significantly reduces the data movement overhead because of the high bandwidth utilization of ring-based broadcast as well as reduced data movement using token-sharding.

Resource Utilization: We use the percentage of time spent on computations to measure the utilization of memory banks. As shown in Figure 11, `Token-TransPIM` has an average 45.8% utilization, which is 1.5× higher than `Layer-TransPIM` (30.8%) because token-based dataflow significantly reduces overhead of data movement. However, `Token-OriginalPIM` and `Token-NBP` provides higher compute utilization, which are 47.7% and 89.5% respectively. This results from the extremely slow computation in PIM-only and NBP-only solutions. Figure 12 shows the average bandwidth utilization, which is the size of reading and writing data divided by the latency. The systems using layer-based dataflow consume more bandwidth than systems with token-based dataflow. For example, `Layer-TransPIM` has up to 1699 GB/s average bandwidth usage while `Token-TransPIM` only has up to 762 GB/s. Considering the overall latency, the result shows that layer-based dataflow requires much more data movements than token-based dataflow. Even though our 8-stack HBM system provides enough bandwidth ($BW_{aggregated} = 8 \times 256 = 2TB/s$), layer-based dataflow may become bandwidth-bound when increasing the workload size or decreasing the system bandwidth. On the hardware side, the usage of data buffer and ring broadcast in TransPIM increases the bandwidth usage of a specific dataflow because of low latency, showing that TransPIM's buffer architecture is always effective.

D. Hardware Customization Exploration

The parallelism of bit-serial adder tree and data buffer size are the two design parameters of ACU. We need

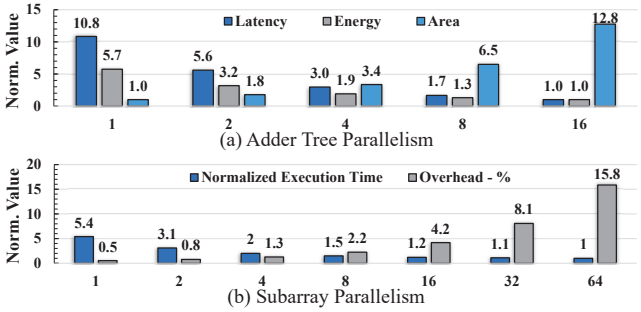


Figure 13. Design space exploration results for ACU.

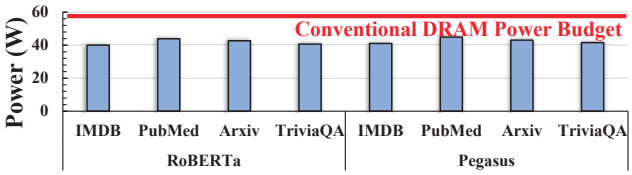


Figure 14. Power consumption of TransPIM on various sequence lengths.

to explore different parameters to find the best tradeoff between additional overhead and resulting performance. We conduct a design space exploration on BERT model by varying the parallelism of adder tree P_{add} from 1 to 16. The results are depicted in Figure 13. The increased adder tree parallelism increases the accessed columns per row activation, thus reducing the number of repeated activated rows during vector reduction. As a result, the latency and energy consumed by vector reduction decrease by at most 10.8× and 5.7×, respectively. Besides, ACU reduces a large part of DRAM access energy by register access energy as shown in Figure 13(a). Finally, we choose $P_{\text{add}} = 4$ as the optimal parallelism for the adder tree in ACU since it keeps a good balance between additional ACU area and performance.

We can enable higher parallelism by simultaneously activating P_{sub} subarrays in a bank, where each subarray contains one independent ACU. However, adding more ACUs in a bank increases the area overhead. Figure 13(b) shows the execution time and area overhead when adding different ACU numbers in a bank. Adding one ACU for each subarray (parallelism = $P_{\text{sub}} = 64$) only increases the performance by 5.4 × while introducing 15.8% area overhead. We choose $P_{\text{sub}} = 8$ to well balance overhead and performance.

E. Power Analysis

We estimate the power consumption of TransPIM for tested workloads, as depicted in Figure 14. Pegasus models on TransPIM dissipate around 2% more power than RoBERTa models under the same sequence length. As the input sequence increases from 128 (IMDB) to 4096 (PubMed), the power of these two models increases about 4W, which is resulted from more computations. Overall, the consumed power of TransPIM is still below the 60W power budget of conventional DRAM system [34]. Thus, TransPIM satisfies the thermal constraints of conventional and TransPIM can be

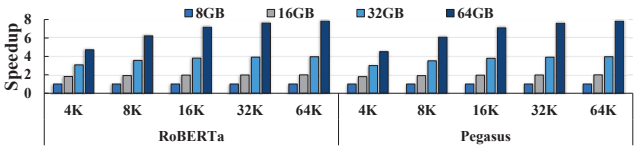


Figure 15. Scalability of TransPIM when increasing the sequence length.

integrated into existing commercial DRAM system without additional modifications in terms of power and cooling.

F. Scalability

As shown in previous work [4], [25], [45], Transformer would become significantly challenging for longer sequences. Memory-based acceleration is promising to provide scalability by simultaneously increasing the memory bandwidth (with low memory access latency) and the compute parallelism. Figure 15 shows the speedup when using more HBM stacks for processing workloads with different sequence lengths. The speedup is averaged across all workloads. The result shows that TransPIM provides good scalability (almost linear) for long sequence workloads which saturate the compute capability of HBM. As the GPU-based solution is bounded by the sequence length due to the limited memory capacity, our experiments indicate that TransPIM is a promising solution to extend the applicability of transformer models for long-sequence applications.

VI. Related Work

Transformer accelerators: A GPU-based serving system and runtime called TurboTransformer is proposed in [13] to process long sequences through maximizing the utilization of computing and memory resources. But the scalability of TurboTransformer is poor since it is unable to support multiple GPUs. In contrast, TransPIM can be easily scaled up by stacking multiple HBM chips to yield larger memory space and support longer sequence lengths. SpAttn [42], A^3 [15], and GOBO [44] are state-of-the-art ASIC processors dedicated for the acceleration of attention module. Both A^3 [15] and SpAttn [42] implement sorting units to prune redundant heads and shrink the memory footprint, thus adapting to the limited on-chip buffer size. SpAttn [42] and GOBO [44] propose low-precision quantization and pipelined architectures to improve the efficiency. Besides, approximate Softmax computation is used in [15]. But GOBO and A^3 are unable to natively support the end-to-end acceleration of the entire Transformer. Moreover, they need to load data from off-chip memory before computation. The off-chip memory bandwidth would become the bottleneck for memory-intensive layers of the Transformer. Instead, TransPIM avoids costly off-chip data transfer by keeping all the data in memory.

PIM accelerators: Various PIM accelerators [16], [17], [27], [29], [37], [40] have been proposed to reduce the overhead of massive data movement as well as support high data parallelism. Newton [16], FIMDRAM [27], and McDRAM [40] adopt the similar near memory architectures and horizontal data organization in memory banks. They cascade bit-parallel arithmetic units to the DRAM bank to perform matrix-vector multiplication. However, the complicated bit-parallel and bulky buffer incur large overhead and decrease memory density [27]. To reduce the overhead of arithmetic units near memory, BFree [37] stores lookup tables that are compatible for computation in memory cells. However, the lookup table requires fine-grained optimization to save the consumed space. Previous in-memory accelerators [12], [17], [29] also optimize the complex reduction. Drisa [29] adds extra shifters in the subarray while NeuralCache [12] relies on the cache I/O peripheral to reorganize data multi-step hierarchical reduction. These two methods either significantly increase the area overhead or introduce large I/O latency. FloatPIM [17] supports reduction by organizing reduction data in a bit-serial way to avoid extra data movement. But this scheme sacrifices parallelism in Transformer which usually has long vectors for reduction. Different from the previous work, TransPIM combines the advantages of in-memory and near-memory computing. The proposed ACU is bit-serial to minimize the overhead of peripheral circuits. Compared to existing PIM accelerators, the proposed PIM-NMC combined computing paradigm provides better efficiency without affecting the memory density. MAT [48] is a PIM-based processing framework for attention-based machine learning models on long-sequence input. It breaks the long-sequence input into segments with various sizes and processes segment in a pipeline manner. However, MAT only targets a single encoder block, different from TransPIM which accelerates the whole Transformer.

VII. Conclusion

In this work, we propose, TransPIM, an end-to-end acceleration for Transformer based on emerging HBM architectures. TransPIM adopts a software-hardware co-design principle to accelerate various Transformer models. As compared to previous accelerators, TransPIM significantly reduces the overhead of data loading by exploiting the data locality in computations associated with input tokens. TransPIM also includes lightweight hardware modifications in HBM to improve the hardware efficiency of computation and data communication. With evaluation on various workloads, TransPIM achieves significantly better performance and energy efficiency than various platforms including GPU, TPU, ASIC, and state-of-the-art memory-based accelerators.

VIII. Acknowledgement

This work was supported in part by CRISP, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program, sponsored by DARPA. This work was also funded by NSF grants (#1730158, #2100237, #1911095, #2112167, #2052809, #1826967).

REFERENCES

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, 2015, pp. 105–117.

[2] M. F. Ali, A. Jaiswal, and K. Roy, "In-memory low-cost bit-serial addition using commodity dram technology," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 1, pp. 155–165, 2019.

[3] K. E. Batcher, "Bit-serial parallel processing systems," *IEEE Computer Architecture Letters*, vol. 31, no. 05, pp. 377–384, 1982.

[4] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[5] G. Bertasius, H. Wang, and L. Torresani, "Is space-time attention all you need for video understanding?" *arXiv preprint arXiv:2102.05095*, 2021.

[6] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 213–229.

[7] K. Chen, S. Li, N. Muralimanohar, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "Cacti-3dd: Architecture-level modeling for 3d die-stacked dram main memory," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2012, pp. 33–38.

[8] Y.-k. Choi, Y. Chi, W. Qiao, N. Samardzic, and J. Cong, "Hbm connect: High-performance hls interconnect for fpga hbm," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 116–126.

[9] A. Cohan, F. Dernoncourt, D. S. Kim, T. Bui, S. Kim, W. Chang, and N. Goharian, "A discourse-aware attention model for abstractive summarization of long documents," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 615–621. [Online]. Available: https://www.aclweb.org/anthology/N18-2097

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[11] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[12] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaaauw, and R. Das, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 383–396.

[13] J. Fang, Y. Yu, C. Zhao, and J. Zhou, "Turbotransformers: an efficient gpu serving system for transformer models," in *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 389–402.

[14] F. Gao, G. Tziantzioulis, and D. Wentzlaff, "Computedram: In-memory compute using off-the-shelf drams," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 100–113.

[15] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "A$\hat{3}$: Accelerating attention mechanisms in neural networks with approximation," in *EEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 328–341.

[16] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.

[17] M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Floatpim: In-memory acceleration of deep neural network training with high precision," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 802–815.

[18] M. Imani, S. Pampana, S. Gupta, M. Zhou, Y. Kim, and T. Rosing, "Dual: Acceleration of clustering algorithms using digital-based processing in-memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 356–371.

[19] "JEDEC Standard JESD235: High Bandwidth Memory (HBM) DRAM," JEDEC Solid State Technology Association, Virginia, USA, Standard, 2013.

[20] H. Jang, J. Kim, J.-E. Jo, J. Lee, and J. Kim, "Mnnfast: A fast and scalable system architecture for memory-augmented neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 250–263.

[21] M. Joshi, E. Choi, D. Weld, and L. Zettlemoyer, "TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1601–1611. [Online]. Available: https://www.aclweb.org/anthology/P17-1147

[22] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *ISCA'17*, 2017, pp. 1–12.

[23] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso, "High bandwidth memory on fpgas: A data analytics perspective," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 1–8.

[24] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[25] N. Kitaev, L. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," in *International Conference on Learning Representations*, 2020.

[26] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 754–768.

[27] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim *et al.*, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 350–352.

[28] C.-C. Lee, C. Hung, C. Cheung, P.-F. Yang, C.-L. Kao, D.-L. Chen, M.-K. Shih, C.-L. C. Chien, Y.-H. Hsiao, L.-C. Chen *et al.*, "An overview of the development of a gpu with integrated hbm on silicon interposer," in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*. IEEE, 2016, pp. 1439–1444.

[29] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017, pp. 288–301.

[30] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," *arXiv preprint arXiv:1506.00019*, 2015.

[31] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[32] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, ser. HLT '11. USA: Association for Computational Linguistics, 2011, p. 142–150.

[33] C. Oh, K. C. Chun, Y. Byun, Y. Kim, S. Kim, Y. Ryu, J. Park, S. Kim, S. Cha, D. Shin, J. Lee, J. Son, B. Ho, S. Cho, B. Kil, S. Ahn, B. Lim, Y. Park, K. Lee, M. Lee, S. Baek, J. Noh, J. Lee, S. Lee, S. Kim, B. Lim, S. Choi, J. Kim, H. Choi, H. Kwon, J. J. Kong, K. Sohn, N. S. Kim, K. Park, and J. Lee, "22.1 a 1.1v 16gb 640gb/s hbm2e dram with a data-bus window-extension technique and a synergetic on-die ecc scheme," in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, 2020, pp. 330–332.

[34] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 41–54.

[35] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[36] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[37] A. K. Ramanathan, G. S. Kalsi, S. Srinivasa, T. M. Chandran, K. R. Pillai, O. J. Omer, V. Narayanan, and S. Subramoney, "Look-up table based energy efficient processing in cache support for neural network acceleration," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 88–101.

[38] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch *et al.*, "Rowclone: Fast and energy-efficient in-dram bulk data copy and initialization," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, 2013, pp. 185–197.

[39] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: In-memory accelerator for bulk bitwise operations using commodity dram technology," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 273–287.

[40] H. Shin, D. Kim, E. Park, S. Park, Y. Park, and S. Yoo, "Mcdram: Low latency and energy-efficient matrix computations in dram," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2613–2622, 2018.

[41] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[42] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," *ArXiv*, vol. abs/2012.09852, 2020.

[43] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 369–383.

[44] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 811–824.

[45] M. Zaheer, G. Guruganesh, A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang *et al.*, "Big bird: Transformers for longer sequences," *arXiv preprint arXiv:2007.14062*, 2020.

[46] J. Zhang, Y. Zhao, M. Saleh, and P. Liu, "Pegasus: Pre-training with extracted gap-sentences for abstractive summarization," in *International Conference on Machine Learning*. PMLR, 2020, pp. 11 328–11 339.

[47] M. Zhou, G. Chen, M. Imani, S. Gupta, W. Zhang, and T. Rosing, "Pim-dl: Boosting dnn inference on digital processing in-memory architectures via data layout optimizations," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021, pp. 1–1.

[48] M. Zhou, Y. Guo, W. Xu, B. Li, K. W. Eliceiri, and T. Rosing, "Mat: Processing in-memory acceleration for long-sequence attention," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 25–30.

[49] M. Zhou, M. Imani, S. Gupta, Y. Kim, and T. Rosing, "Gram: Graph processing in a reram-based computational memory," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 591–596. [Online]. Available: https://doi.org/10.1145/3287624.3287711

[50] M. Zhou, L. Wu, M. Li, N. Moshiri, K. Skadron, and T. Rosing, "Ultra efficient acceleration for de novo genome assembly via near-memory computing," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2021, pp. 199–212.

[51] M. Zhu, Y. Zhuo, C. Wang, W. Chen, and Y. Xie, "Performance evaluation and optimization of hbm-enabled gpu for data-intensive applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 5, pp. 831–840, 2018.