AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference

Jaehyun Park* Seoul National University South Korea jhpark@scale.snu.ac.kr

Jaewan Choi* Seoul National University South Korea jwchoi@scale.snu.ac.kr

Kwanhee Kyung Seoul National University South Korea kwanhee5@snu.ac.kr

Michael Jaemin Kim Seoul National University South Korea michael604@snu.ac.kr

Yongsuk Kwon Seoul National University South Korea happyysyy@snu.ac.kr

Nam Sung Kim University of Illinois Urbana-Champaign United States nskim@illinois.edu

Jung Ho Ahn Seoul National University South Korea gajh@snu.ac.kr

Abstract

The Transformer-based generative model (TbGM), comprising summarization (Sum) and generation (Gen) stages, has demonstrated unprecedented generative performance across a wide range of applications. However, it also demands immense amounts of compute and memory resources. Especially, the Gen stages, consisting of the *attention* and *fully-connected* (FC) layers, dominate the overall execution time. Meanwhile, we reveal that the conventional system with GPUs used for TbGM inference cannot efficiently execute the attention layer, even with batching, due to various constraints. To address this inefficiency, we first propose AttAcc, a processing-in-memory (PIM) architecture for efficient execution of the attention layer. Subsequently, for the end-to-end acceleration of TbGM inference, we propose a novel heterogeneous system architecture and optimizations that strategically use xPU and PIM together. It leverages the high memory bandwidth of AttAcc for the attention layer and the powerful compute capability of the conventional system for the FC layer. Lastly, we demonstrate that our GPU-PIM system outperforms the conventional system with the same memory capacity, improving performance and energy efficiency of running a 175B TbGM by up to 2.81× and 2.67×, respectively.

CCS Concepts: • **Computer systems organization → Parallel architectures**.

Keywords: Processing-in-Memory, Transformer-based Generative Model, DRAM

ACM Reference Format:
Jaehyun Park, Jaewan Choi, Kwanhee Kyung, Michael Jaemin Kim, Yongsuk Kwon, Nam Sung Kim, and Jung Ho Ahn. 2024. AttAcc! Unleashing the Power of PIM for Batched Transformer-based Generative Model Inference. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3620665.3640422

*Both authors contributed equally to the paper.

1 Introduction

Since its introduction, the Transformer-based generative model (TbGM) has showcased exceptional output quality across diverse application types. These include not only tasks in the field of natural language processing such as chatbot, text summarization, and code generation, but also image, video, and speech processing domains. TbGM inference consists of one summarization stage (Sum stage) and recurring generation stages (Gen stages), both of which are composed of multiple fully-connected (FC) and attention layers. Due to its sequential nature, the Gen stage generally dominates the total execution time. TbGM takes an input sequence of length L_{in}, which has been increasing (see Table 1). Each Sum/Gen stage produces an output token, ultimately generating an output sequence of length L_{out} until termination.

Table 1. Model size and maximum input sequence trends GPT models assuming FP16 weights.

Model	GPT-1 [51]	GPT-2 [52]	GPT-3 [5]	GPT-4 [44]
Max. model size	0.21GB	2.8GB	326GB	-
Max. input seq. len.	512	1,024	2,048	32,768

Despite its remarkable capabilities, TbGMs demand unprecedented amounts of compute and memory resources. For example, GPT-3 [5], a representative TbGM, has up to 175 billion parameters and requires 1,475 TFLOPs of computation (*i.e.*, more than 300,000× of ResNet-50) when (L_{in}, L_{out}) is (2,048, 2,048).[1] In such a case, both the FC and attention layers in the Gen stage end up demanding more memory bandwidth than what the current high-end GPUs can provide. That is, the conventional system with such GPUs becomes inefficient for TbGM inference. For example, running GPT-3 with (L_{in}, L_{out}) of (2,048, 2,048) on an NVIDIA DGX A100 system [42] with 640GB of HBM3 and 26.8TB/s of memory bandwidth (henceforth *baseline*) will leave its compute unit utilization below 1%.

Batching to process multiple input sequences together can be a potential solution to inference throughput, but it is effective only for the FC layer, not the attention layer. Specifically, batching can improve the utilization of the system's compute and memory resources for the FC layer by reusing the weight matrices and increasing arithmetic intensity, quantified by the Operation per Byte (Op/B). For example, with a batch size of 256 and unlimited memory capacity for the *baseline*, the GPU utilization reaches 71%. Besides, its throughput increases by 88× while maintaining nearly the same latency. However, the Key and Value (KV) matrices of the attention layer are *unique* per (inference) request. That is, batching can neither reuse the KV matrices nor improve the throughput of processing the attention layer.

Moreover, the attention layer even limits the maximum batch size and impacts the FC layer throughput due to two critical constraints: memory capacity and service level objective (SLO). (1) The memory capacity required to store KV matrices can be prohibitively high. For instance, the KV matrices of each request can be up to 18GB for an (L_{in}, L_{out}) of (2,048, 2,048), which increases proportionally to the batch size. Thus, for the *baseline* system with 640GB of memory, the maximum batch size is 18. (2) Even if the memory capacity constraint is resolved, the SLO requirement becomes another limiting factor. As batching does not improve the throughput of the attention layer (§3.2), a larger batch leads to a longer processing time and thus violation of a given SLO constraint. For example, we can hypothesize an unrealistic version of *baseline* with 5,000GB of memory, which seems to enable the batch size of 256. Nonetheless, when the SLO is 50ms, in line with prior work [49, 66], the maximum batch size can be merely 27.

[1]Considering that the maximum sequence length for the recently released ChatGPT even surpasses 4,096 [44], we set L_{out} to a maximum of 2,048 when L_{in} is 2,048.

To address such inefficiency, we first propose a processing-in-memory (PIM) architecture accelerating memory-bound attention layers, dubbed AttAcc (**Att**ention **Acc**elerators). Specifically, we start with an extensive design space exploration to design the microarchitecture of AttAcc. This includes where to place processing units within the DRAM-based PIM (*e.g.*, buffer die, bank-group, and bank) for both GEMV and softmax operations, while meticulously considering bandwidth requirement, area overhead, and power budgets. Then, we devise an efficient data-mapping methodology for AttAcc after evaluating tradeoffs, such as between bandwidth consumption and exploitable degree of parallelism, among various data mapping schemes.

Second, for the end-to-end acceleration of TbGM inference, we propose a heterogeneous system architecture that strategically processes the memory-bound attention layers with AttAcc, while efficiently handling compute-bound batched FC layers with xPUs (*e.g.*, GPUs or TPUs). This unique optimization opportunities maximize the utilization of both xPUs and AttAcc*s*. AttAcc alone has already reduced the execution time of the attention layer. This allows us to increase the maximum batch size under the SLO constraint, and thus the throughput of processing the FC layer. Additionally, we propose to have xPUs and AttAcc*s* pipeline execution of operations in the FC and attention layers. This improves the utilization of both xPUs and AttAcc*s*, decreases the total execution time of both the attention and FC layers for each batch, and thus further increases the maximum batch size.

Our evaluation shows that the GPU-PIM heterogeneous system with these optimizations can offer up to 2.81× higher performance and 2.67× lower energy consumption than the conventional GPU system when both the systems are populated with 1,280GB of main memory.

The key contributions of this paper are as follows:

- We discover that the attention layer poses a constraint on the batch size in executing TbGM with conventional platforms (e.g., GPUs) due to the SLO and memory capacity requirements.
- We propose a heterogeneous system architecture with the conventional computing platform for the batched FC layer and AttAcc*s* for the attention layer, leveraging PIM architecture.
- We propose efficient pipelining and co-processing optimizations that maximize the utilization of AttAcc*s*.

2 Background

2.1 GPT Model

In this paper, we focus on GPT [5], a representative TbGM. The GPT architecture is based on the decoder of Transformer [61] (see Figure 1). In a GPT model, input tokens

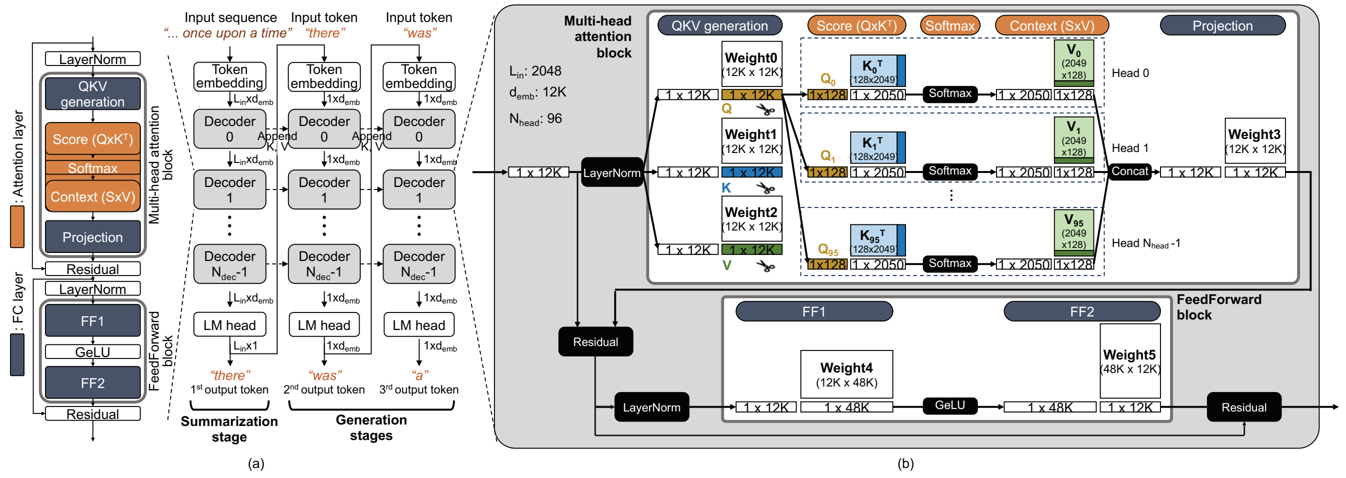

Figure 1. (a) The inference process of GPT and (b) the detailed process of one decoder in a Gen stage for GPT-3 175B.

are first converted into embedding vectors through token embedding [61]. These vectors are cascaded through N_{dec} decoders, each with different pre-trained weights, to generate an output embedding vector. The output vector is converted to an output token through a language model (LM) head.

A decoder consists of a multi-head attention block and a feedforward (FF) block. Both blocks are accompanied by layer normalization (layernorm), a type of normalization performed on the embedding vector, and residual, element-wise addition. The multi-head attention block is composed of three layers in the following order: QKV generation layer, attention layer, and projection layer. The QKV generation and projection layers are fully connected (FC) layers for generating query (Q), key (K), and value (V) data that contain the information of each input token and for projecting the outputs to the attention layer. The attention layer calculates the contextual representation between tokens (Figure 1(b)). It is divided into multiple independent heads, which can be executed in parallel. The Q, K, and V data are equally divided into N_{head} heads (Q_i, K_i, and V_i for i = 0, 1, ..., $N_{head}-1$). Each head sequentially performs an inner product (score operation) on Q_i and K_i, a softmax operation on the inner product result, and another inner product (context operation) with V_i on the softmax result. The outputs of the heads are concatenated and used as input for the projection layer. The feedforward block consists of two FC layers (FF1 and FF2) and a Gaussian error linear unit (GELU) layer.

2.2 GPT Inference Process

GPT infers a request by going through a summarization (Sum) stage that understands the context of an input sequence, followed by multiple generation (Gen) stages, each generating a new token in an autoregressive manner. Each stage proceeds sequentially and reuses the same pre-trained weights. The primary operations in the Sum stage are the

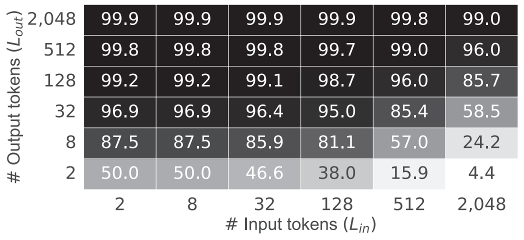

Figure 2. Percentage of the Gen stage time in total execution time over various input and output tokens on GPT-3 175B with batch size 1.

general matrix-matrix multiplication (GEMM). In the Sum stage, each decoder receives a matrix of $L_{in} \times d_{emb}$ (embedding dimension) as an input, processes it through the FC and attention layers, and outputs a matrix of the same size.

The Gen stage proceeds similarly to the Sum stage but with two major differences. First, a Gen stage mainly performs general matrix-vector multiplication (GEMV) operations. It takes a single token, generated by the previous stage, as an input, and the input of each decoder is also a vector. Second, the attention layer operates on the Q vector for an input token of the current stage with the KV matrices that aggregate the input tokens of the Sum stage with the generated tokens up to the previous Gen stage. That is, each Gen stage appends newly created KV vectors of an input token to the KV matrices transferred from the previous stage. When L is defined as the sum of L_{in} and the number of tokens generated up to the current stage, the dimensions of the Q vector and the KV matrices are $1 \times d_{emb}$ and $L \times d_{emb}$, respectively. The output token of a Gen stage becomes the input of the next Gen stage, being repeated until an end-of-sequence token is generated.

The Gen stages typically overwhelm the Sum stage in execution time due to their sequential nature of reading the entire pre-trained weights per generated (output) token. Figure 2 shows the percentage of time taken by the Gen stages over the total execution time for various combinations of L_{in} and L_{out} in the GPT-3 175B model. We performed the experiments with our in-house simulator assuming the peak FLOPS (2.5 PFLOPS) and maximum memory bandwidth (26.8 TB/s) of the NVIDIA DGX A100 with 640GB of HBM3 (DGX).[2] When both L_{in} and L_{out} are 32, the Gen stages consume more than 96% of the total execution time. Even if L_{in} is 2,048, the longest Sum stage in our configuration, and L_{out} is 128, the percentage of Gen stages also exceeds 85%. This trend can also be confirmed by prior works [23, 69] studying GPT-2. We focus on the Gen stages hereafter.

3 Benefits and Limits of Batching for TbGM

In this section, we first analyze the benefits of batching for GPT on the DGX system, a representative platform for serving large TbGMs. Then, we identify the limitations of the DGX system on large batch sizes while processing the attention layer. We use the latest batching technique (*i.e.*, iteration-level scheduling [66]), where a new request is added to the batch whenever a request is completed. Such a technique can eliminate the idle time of early completed requests even when the number of stages (L_{out}) is different for each request.

3.1 Benefits

For the FC layer of TbGM inference, batching increases the reusability of the weight matrices; it improves both the throughput and energy efficiency. Specifically, batching transforms the FC layer from a GEMV to GEMM operation through weight sharing. Once the weights are read from the off-chip memory, all the requests in a batch can reuse the weights, improving the arithmetic intensity (see Figure 3) or operation per byte (Op/B). Thus, even when the FC layer of multiple requests in a batch is processed simultaneously, only the throughput increases while the execution time stays nearly the same. Moreover, off-chip memory access for the weight matrices is also saved, decreasing the overall energy consumption. Figure 4(a) and (b) represent the throughput (generated tokens per second) and energy consumption. For all three configurations of L_{out} with L_{in} of 2,048, an increase in batch size leads to an improvement in throughput and a reduction in energy consumption.

3.2 Limitations

Despite the benefits of batching for the FC layer, the attention layer gives new challenges as the batch size (the number of requests per batch) increases. As the size of KV matrices scales with batch size, it can exceed the memory capacity of conventional platforms. Notably, the attention layer uses distinct, request-specific KV matrices whose sizes also grow

[2] All of our analyses and evaluations are based on a system where the GPU memory is replaced with HBM3 in the original DGX-A100.

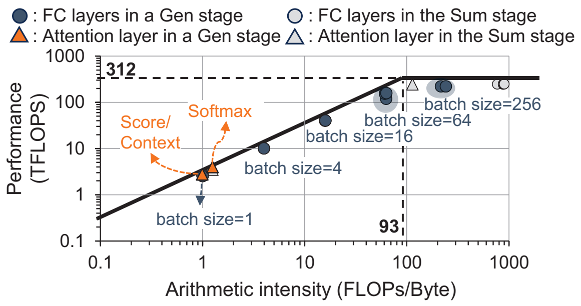

Figure 3. Roofline model of the *baseline* for the Sum stage and the first Gen stage of GPT-3 175B when L_{in} is 2,048. The dots for the attention layer are located at the same point regardless of the batch size.

proportionally to sequence length. The number of elements of KV matrices is $2\times(L_{in}+L_{out})\times d_{emb}$ per decoder, resulting in $2\times N_{dec}\times(L_{in}+L_{out})\times d_{emb}$ per request. For example, the size of KV matrices in the GPT-3 175B model using FP16 with (L_{in}, L_{out}) of (2,048, 2,048), is 18GB per request; thus, the required memory capacity for a batch size of 64 is about 1.5TB, including 326GB of weights and 1,152GB of KV matrices, which is much larger than DGX's 640GB memory capacity (see Figure 4(a)).

Even if the memory capacity constraint is resolved, the attention layer limits the overall batch size under the SLO constraint. As batching does not improve the throughput of the attention layer, the execution time for processing a batch increases with the batch size. That is, when the SLO is fixed, the maximum batch size is limited due to the attention layer. For example, Figure 4(c) shows that the attention layer can account for more than half of the total execution time when the batch size is 64. In fact, the output token generation latency of a batch size of 64 rises to 80ms, which violates a reasonably set 50ms SLO of GPT-3. Batch sizes close to 16 are necessary to adhere to this latency constraint.

Further, the attention layer has a low arithmetic intensity regardless of batch size (see Figure 3). The primary operation of the attention layer in the Gen stage is the GEMV of the score and context operations, exhibiting a low Op/B (~1). Unlike the FC layer, the attention layer still has memory-intensive GEMV operations even after batching because each request uses unique KV matrices, as explained in §2. Thus, the computing units of DGX are mostly idle for the attention layer, even with batching. As shown in Figure 4(c), the GPU utilization increases due to the improved arithmetic intensity of the FC layers. However, even when the batch size reaches 256, the utilization remains below 20% because of the substantial time spent on the attention layer. To sum up, the attention layer does not benefit from large batching and also constrains the batch size on conventional platforms, which in return even degrades the throughput of the FC layer.

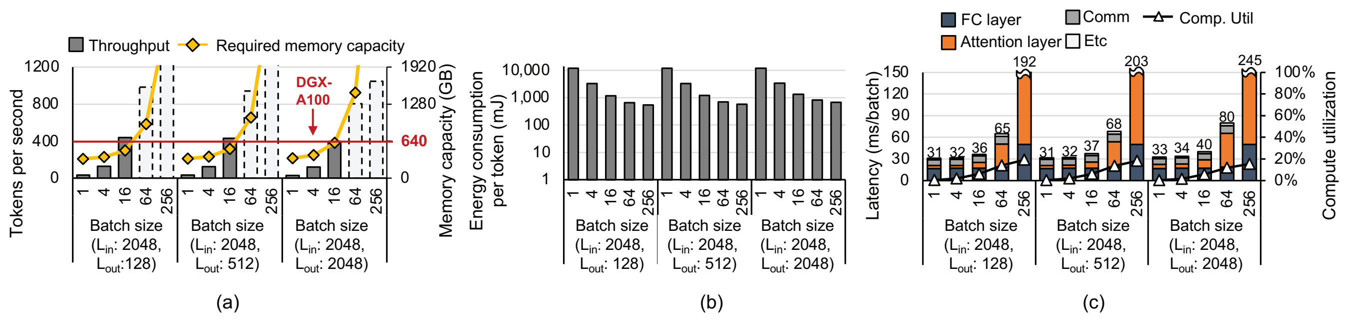

Figure 4. (a) Throughput and required memory capacity, (b) energy consumption per output token, and (c) the Gen stage time breakdown and GPU compute utilization for various batch sizes with L_{in} of 2,048 and L_{out} of 128, 512, or 2,048 when running the GPT-3 175B model on DGX assuming unlimited memory capacity. The dotted bars in (a) indicate the throughput with a batch size not available with DGX A100 due to memory capacity limitation.

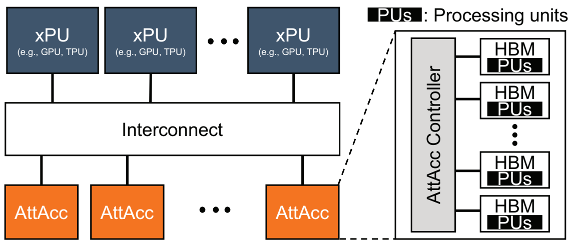

Figure 5. The overall heterogeneous computing platform with xPUs and AttAccs.

3.3 PIM for Attention Layers

To overcome these limitations, we may scale out the conventional system to increase the total memory bandwidth. While scaling out can increase the attention layer throughput, such an approach is less efficient due to low utilization of compute resources and large amounts of off-chip data movement. Instead, we propose to use a processing-in-memory (PIM) architecture that exposes the abundant internal bandwidth to processing units (PUs) closer to the memory.

The acceleration of the attention layer is a compelling application for PIM architecture because of two reasons. First, the KV matrices of the attention layer can be written to DRAM by the Sum stage only once, using the DRAM *external (off-chip) bandwidth*. Then, it can be read multiple times by the Gen stage, leveraging the DRAM *internal (on-chip) bandwidth*. Although KV matrices must be updated for each Gen stage, the appended vector is small. Second, the GEMV operation is a reduction operation. That is, the sizes of input and output data that traverse the off-chip interface are small, while the large KV matrices stay in DRAM without consuming the external bandwidth. The N_{head} GEMV operations of the score and context operations have (input, K/V matrix, output) size of $(1\times d_{emb}, L\times d_{emb}, N_{head}\times L)$ and $(N_{head}\times L, L\times d_{emb}, 1\times d_{emb})$, respectively. Thus, the ratio of the input and output data over KV matrices, *i.e.*, the ratio of the external to internal bandwidth consumption, corresponds to $(d_{emb}+N_{head}\times L)/(L\times d_{emb})$. This ratio reaches up to 1/128 for GPT-3 with 175B weights, having a d_{emb} of 12,228, an N_{head} of 96, and an L greater than or equal to 2,048.

4 TbGM Accelerator Architecture

In this section, we propose a heterogeneous computing platform to accelerate TbGM, which ameliorates the inefficiencies of conventional computing platforms (see Figure 5). High-performance compute units (xPUs) such as GPUs or TPUs [28] handle batched FC layers. AttAcc, leveraging the high bandwidth memory (HBM)-based PIM architecture, aims to accelerate the attention layer. Commercial high-bandwidth interconnects, such as PCIe, NVLink, or CXL [11], bridge AttAccs with the xPUs. We perform two design space explorations regarding the architecture of AttAcc; i) where to put the processing units and ii) how to lay out the data. The implementation details of AttAcc such as processing unit microarchitecture, control unit operation, and programming model are provided in §5.

4.1 Architecting PIM for AttAcc

We explore the design space of AttAcc based on the 8-Hi HBM3 [26]. 8-Hi HBM3 (hereafter HBM) consists of eight DRAM dies and a buffer die that are 3D stacked using through silicon vias (TSVs) (see Figure 6). A group of four DRAM dies constitutes a rank. A DRAM die is composed of eight pseudo channels (pCHs) that operate independently, and each pCH comprises sixteen banks grouped into four bank groups (BGs). When a read command is issued to HBM, the data from DRAM cells travels to the buffer die through the local I/O of the bank, BG bus, global bus (GBUS), and TSVs, and then moves to the outside of HBM through the PHY.

AttAcc has two different processing units of GEMV unit and softmax unit. Upon the aforementioned basic HBM structure, we quantify the performance, energy consumption, and area overhead tradeoffs of placing the two processing units at different memory hierarchies.

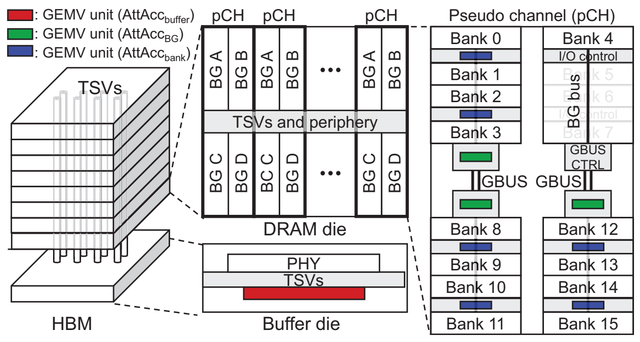

Figure 6. The HBM architecture and GEMV unit placement for AttAcc$_{buffer}$, AttAcc$_{BG}$, and AttAcc$_{bank}$ in HBM.

GEMV Unit Placement: Where to locate the processing units for GEMV (GEMV units), whether i) per pseudo-channel at the buffer die (AttAcc$_{buffer}$), ii) per bank group (AttAcc$_{BG}$), or iii) per bank (AttAcc$_{bank}$) at the DRAM die, significantly impacts performance, energy consumption, and area overhead (see Figure 6 and 7). Placing the GEMV unit deeper in the HBM hierarchy increases the aggregate internal bandwidth and reduces energy consumption due to shorter data travel distances. However, it leads to higher area overhead as more GEMV units are employed and uses a DRAM fabrication process instead of a logic process. There is also an upper limit to the exploitable internal bandwidth due to HBM's power constraint. We calculate the HBM power budget using the loop pattern of all-bank interleave read current (IDD7) defined by JEDEC [26]. We assume that the buffer die of HBM leverages a logic process that offers better performance and area efficiency than a DRAM process [30, 48].

First, AttAcc$_{buffer}$ allocates a GEMV unit per pCH at the buffer die, with an accumulator that aggregates the results from different pCHs. This allows for full utilization of the pCH data path (TSVs). AttAcc$_{buffer}$ can exploit the logic process for the GEMV unit, allowing for a small area overhead. However, as the HBM I/O bandwidth is identical to the aggregate bandwidth the GEMV units can exploit, the performance benefit from PIM is nonexistent. Further, because data has to traverse all the way from DRAM cells to the buffer die, there exist little energy benefits.

Second, AttAcc$_{BG}$ allocates GEMV units at the GBUS controller (GBUS CTRL) on a DRAM die, and puts accumulators on a buffer die to gather the data from different bank groups. AttAcc$_{BG}$ can exploit the internal bandwidth, which is higher than the external bandwidth by a factor of (#_of_ranks × #_of_BGs_per_pCH). Considering that the GEMV unit in AttAcc$_{BG}$ can read data per tCCDL (delay between two consecutive read commands to the same BG), which is longer than tCCDS [26], AttAcc$_{BG}$ can achieve 4× higher internal bandwidth than external bandwidth for HBM. The energy overhead also decreases as the amount of data that traverses TSVs and I/O PHY is reduced. Still, as the BGs that operate concurrently put pressure on the power budget, there is a limitation on the number of GEMV units that can simultaneously run (*e.g.*, 6 GEMV units per pCH for HBM3). Moreover, because GEMV units are populated more and implemented using a DRAM process, the area overhead is aggravated.

Lastly, AttAcc$_{bank}$ allocates the GEMV unit per bank close to the column decoder, and the hierarchical accumulators gather data at the GBUS CTRL per BG, and at the buffer die per pCH. AttAcc$_{bank}$ can populate more GEMV units than AttAcc$_{BG}$ by the amount proportional to the number of banks per BG (*e.g.*, 4). However, AttAcc$_{bank}$ is more prone to the performance penalty due to the time for activating/precharging DRAM rows compared to AttAcc$_{BG}$. It is because each GEMV unit reads data only from one bank; thus, time for activating/precharging rows cannot be hidden across multiple banks within a single GEMV unit. Meanwhile, the number of GEMV units that can concurrently operate within the HBM power budget increases compared to AttAcc$_{BG}$ because data traverses a shorter distance, consuming less power (see Figure 7(b)). For example, 18 GEMV units with an internal bandwidth of 9× the external bandwidth can run simultaneously per pCH for HBM3 under the same power constraint. In this case, the time to activate/precharge a row in a bank can be hidden by the time to read data from other banks that are not operating simultaneously. The area overhead increases, as more GEMV units are employed, which again uses a DRAM fabrication process.

Among the three options, we choose AttAcc$_{bank}$ as our choice taking all three performance, energy, and area overhead into account, under the power constraint (see Figure 7). While the experimental details are provided in §7.1, Figure 7(d) shows the energy-delay-area product (EDAP) of each design. Across all tested TbGMs (LLAMA, GPT-3, and MT-NLG), there exists ample parallelism all GEMV units can exploit, resulting in a high performance and energy advantage of AttAcc$_{bank}$. As the area overhead increase is limited to 10.84%, AttAcc$_{bank}$ demonstrates superiority.

Softmax Unit Placement: We choose to place the softmax unit at the buffer die of each HBM for the following reasons. First, locating the softmax unit at the buffer die eliminates unnecessary data transfers. As the softmax unit collects all the output data from GEMV for score operation (GEMV$_{score}$) and feeds the input data to GEMV for context operation (GEMV$_{context}$), positioning them deeper in the hierarchy results in unnecessary and complex transfer between BGs or banks of a different die.

Second, placing the softmax unit deeper in the HBM hierarchy is overkill in the total number of units and provisioned bandwidth. The maximum number of softmax units is the total number of heads because a softmax unit should handle at least one head as a whole to eliminate unnecessary data movements. However, the total number of BGs or banks is typically higher than the number of heads. For example, the maximum number of softmax units is 4,800 for 96 heads and

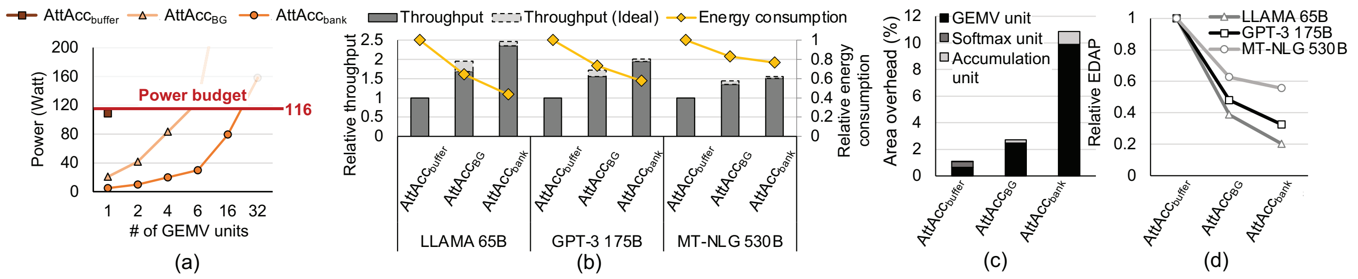

Figure 7. (a) Peak power, (b) relative throughput and energy consumption, (c) area overhead, and (d) relative EDAP for AttAcc$_{\text{buffer}}$, AttAcc$_{\text{BG}}$, and AttAcc$_{\text{bank}}$. Throughput and energy consumption are for the GPT-3 175B with L_{in}/L_{out} of 2,048.

a batch size of 50 (GPT-3 175B with 2,048 L and 1,280GB of memory). By contrast, the total number of banks operating in parallel for AttAcc$_{\text{bank}}$ with 40 8-Hi HBM3 is 40,960. Moreover, in terms of required bandwidth, the ratio of data needed for softmax and GEMV is N_{head}/d_{emb} (*e.g.*, 1/128) as explained in §3.3. Because the buffer die of 8-Hi HBM3 can provision 1/9 of the AttAcc$_{\text{bank}}$ aggregate internal bandwidth, it is enough to place the softmax units on the buffer die.

Lastly, using a DRAM process for the softmax unit can be burdensome due to its relatively complex processing units (Figure 9(c)) and the requirement for large SRAM buffers to store intermediate vectors. Assuming that the DRAM process is 10× less dense than the logic process [12], the area of the softmax unit would reach 63.4mm^2 for a 1z-nm DRAM process (details in §5.1).

4.2 Exploring Data Mapping for AttAcc

Mapping of the KV matrices can be decided hierarchically at multiple levels; i) HBM level, ii) pCH, bank group, and bank level, and iii) GEMV unit multiplier level. First, we map each head to one HBM, while multiple heads can be mapped to a single HBM regardless of which request it belongs to (see Figure 8(a)). This mapping is possible because, as long as the KV matrices that belong to the head i (henceforth K_i^T and V_i) and the appropriate Q vector are located in that HBM, GEMV$_{\text{score}}$/GEMV$_{\text{context}}$ of the head can be executed independently. Such head-wise mapping eliminates unnecessary data transfer between the HBMs. As the size of each K_i^T/V_i is different per request, dependent on L, load imbalance between HBMs may occur. Each head of a new request is greedily allocated to the HBMs in a way to minimize such imbalance. The allocation is conducted at the Sum stage.

Then, at every pCH, bank group, and bank, each K_i^T/V_i can be partitioned either row-wise or column-wise. As shown in Figure 8(b), two methods impact the amount of data transfer and the exploitable degree of parallelism. First, row-wise partitioning reduces the amount of input vector data transfer while column-wise partitioning minimizes the output vector data transfer. As broadcasting the identical input vector to multiple bank groups and banks is possible via a multi-drop bus of the current DRAM structure [22], reducing the output vector data transfer is more reasonable. Second, the degree of parallelism depends on the number of rows/columns for row-wise/column-wise partitioning, respectively. Considering large L, the shape of V_i and K_i^T is $L{\times}d_{head}$ (dimension of a head) and $d_{head}{\times}L$, respectively. d_{head} is typically not large enough to provide the degree of parallelism to fully utilize all GEMV units. For example, the number of columns in V_i is d_{head} (128 for GPT-3 175B), which is much smaller than the 1,024 GEMV units in an 8-Hi HBM3 of AttAcc. In such a case, we can use row-wise partitioning instead, having L degree of parallelism. Partitioning types can be chosen differently at each level of pCH, bank group, and bank.

Lastly, at the GEMV unit multiplier level, we choose to use row-wise partitioning for K_i^T and column-wise partitioning for V_i (see Figure 8(c)). This strategy is to prevent the possible load imbalance between the multipliers caused by the KV vectors appending on the KV matrices at every Gen stage. For example, if the K_i^T matrix is partitioned column-wise, the appended $1 \times d_{head}$ column vector must be sequentially processed by one multiplier, resulting in a load imbalance with the other multipliers.

In summary, we choose i) the row-/column-wise partitioning for K_i^T/V_i at the multiplier level, ii) for (pCH, BG, bank), (column, column, row)-wise partitioning for GEMV$_{\text{score}}$/K_i^T and (row, row, column)-wise partitioning for GEMV$_{\text{context}}$/V_i, and iii) head-wise partitioning at the HBM level.

5 AttAcc Implementation

In this section, we provide the microarchitectural details of AttAcc components, *e.g.*, the GEMV unit, softmax unit, accumulator, and AttAcc controller. Also, we introduce the programming model of AttAcc and the detailed execution flow of TbGM inference under our heterogeneous computing platform composed of xPUs and AttAccs.

5.1 AttAcc Component Microarchitecture

GEMV Unit: The GEMV unit reads data from DRAM cells and processes score and context operations. Each GEMV unit consists of 16 FP16 multipliers, 16 FP16 adders, double-buffered 16 256-bit buffers that can store the input vectors, and a control unit. As discussed in §4.2, the multipliers in the

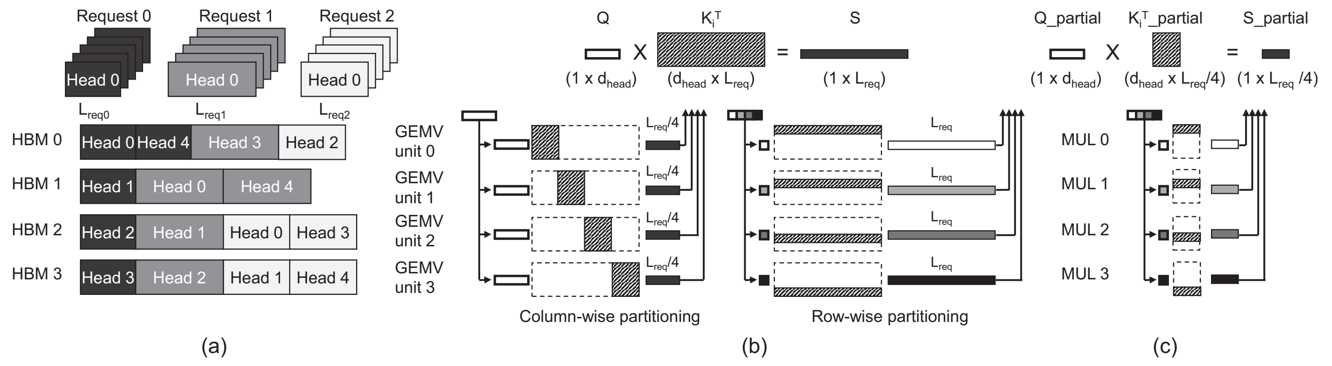

Figure 8. Methodologies to map K_i^T for GEMV$_{\text{score}}$ to (a) HBMs, (b) pCHs/BGs/banks, (c) multipliers in GEMV unit. Because V_i for GEMV$_{\text{context}}$ is in the transposed form of K_i^T, mapping for V_i applies the reverse of K_i^T to pCH/BG/bank/multiplier levels.

GEMV unit can take either row- or column-wise partitioning for K_i^T and V_i. The adders operate as an adder tree for row-wise partitioning and as an accumulator for column-wise partitioning. We adopt multiplexers at the input of each adder, allowing the time-multiplexing of the adders.

Softmax Unit: The softmax unit in a buffer die consists of 256 FP32 exponent units, 256 FP32 adders, 256 FP32 multipliers, a comparator tree, an adder tree, and an FP32 divider (see Figure 9(c)). The number of arithmetic units is configured to match the throughput of GEMV units. The softmax unit also has a 512KB buffer that can store the input and output of the softmax operation. The softmax unit operates in three pipeline stages; maximum value calculation, exponent calculation, and normalization.

Accumulator: In the case of row-wise partitioning at the pCH/BG/bank, accumulators are employed to aggregate and reduce the partial results from different GEMV units. The accumulators are located hierarchically between the GEMV and the softmax units, per BG on the DRAM die and per pCH on the buffer die. For example, when row-wise partitioning is employed for the V_i matrix at the bank, the per-BG accumulators reduce the partial result. Such a reduction operation at the accumulator decreases the amount of overall data transfer. The accumulator at each hierarchy is simply bypassed when a column-wise partitioning is used.

AttAcc Controller: The AttAcc controller consists of an I/O module, a direct memory access (DMA) engine, an instruction queue and decoder, config memory, and multiple HBM controllers (see Figure 9(a)). AttAcc receives instructions for the attention layer (henceforth Att_inst), where each has a one-to-one corresponding AttAcc API function. Att_insts are decoded to trigger i) operations for an attention layer through HBM controllers, ii) data transfers between the AttAcc and xPUs, or iii) modify the config memory. The HBM controller, one per HBM, generates DRAM or PIM commands for each channel and performs score, softmax, and context operations for each request. The data transfer is supported by the DMA engine. Config memory stores N_{head}, d_{head}, maximum L, information about KV matrix mapping (row-wise or column-wise) at each memory node level, batch size, and L of each request within the batch.

PIM commands issued to the HBM include the ones for metadata setup and data movement. `PIM_SET_CONFIG` writes the partitioning information of the KV matrices to the GEMV units. `PIM_ACT_AB` performs row activation of the same address in all banks, similar to prior works [36, 37]. `PIM_MAC_AB` executes the multiply-accumulate (MAC) operations in all banks while reading necessary data from DRAM cells and GEMV buffers. `PIM_SFM` executes the softmax operation including the computations such as finding maximum, subtraction, and exponential. `PIM_WR_GB` writes data to a GEMV buffer. `PIM_MV_GB` moves the output data from the GEMV unit to the softmax buffer. Similarly, `PIM_MV_SB` moves the result from the softmax unit to the GEMV buffer. `PIM_RD_SB` reads the final results of a context operation from a softmax buffer. All PIM commands are implemented as RFU (reserved for future use) commands. They are issued through a standard HBM command path, similar to DRAM commands.

5.2 Programming Model and Execution Flow

Programming Model: AttAcc employs a heterogeneous computing programming model, similar to CUDA [40] and OpenCL [60]. The host offloads the attention layers of a Gen stage to AttAcc using AttAcc API with the support of AttAcc run-time device driver. The control registers and config memory of AttAcc are exposed to the host as memory-mapped I/O. The rest of the HBM memory of AttAcc can only be accessed by others through the AttAcc API functions with the support of the driver and DMA engine.

Each AttAcc API function sends a corresponding Att_inst to an AttAcc controller. To be more specific for each AttAcc API function, *AttAcc::SetModel* and *AttAcc::UpdateRequest* initialize the config memory. *AttAcc::MemCopy* moves data between AttAcc and xPUs through DMA with the support of the AttAcc driver. The specific method of communication

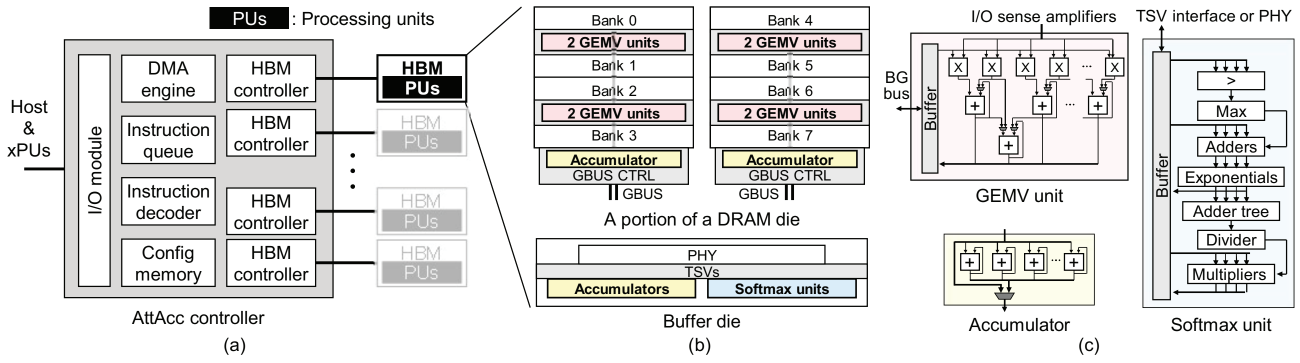

Figure 9. (a) The architecture of AttAcc, (b) its processing units in HBM, and (c) the details of the processing units.

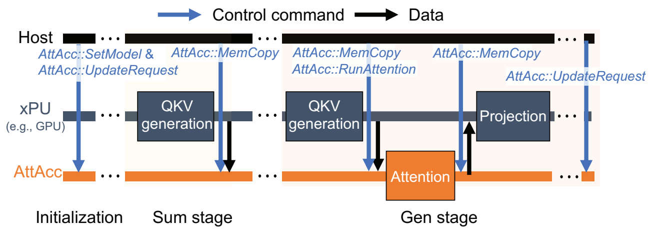

Figure 10. The execution flow of TbGM inference in a heterogeneous computing platform including xPUs and AttAccs.

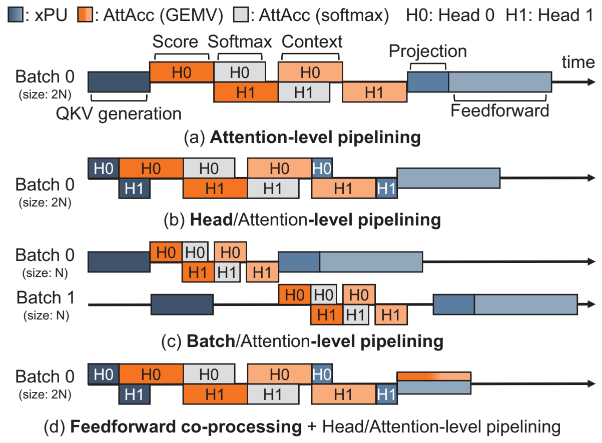
: xPU : AttAcc (GEMV) : AttAcc (softmax) H0: Head 0 H1: Head 1
Score Softmax Context
Projection
time
Batch 0 (size: 2N)
QKV generation
Feedforward
(a) Attention-level pipelining
(b) Head/Attention-level pipelining
Batch 1 (size: N)
(c) Batch/Attention-level pipelining
(d) Feedforward co-processing + Head/Attention-level pipelining

Figure 11. The timing diagrams for processing Gen stages in the heterogeneous computing platform consisting of xPUs and AttAccs with attention-level, head-level, and head-level pipelining and feedforward co-processing.

between the two can vary depending on the type of interconnect (*e.g.*, P2P [56] for PCIe). *AttAcc::RunAttention*, one for each attention head, instructs the AttAcc controller to initiate the attention operation.

Execution Flow: The execution flow of TbGM inference in a heterogeneous platform including xPUs and AttAccs is as follows (Figure 10). Initially, the host sends model configuration and request information to the config memory in the AttAcc controller through *AttAcc::SetModel* and *AttAcc::UpdateRequest*. In the Sum stage of each head, the host offloads all operations to the xPUs and executes *AttAcc::MemCopy* to transfer KV matrices to AttAccs whenever they are generated in the xPUs. *AttAcc::Memcopy* accompanies arguments including the xPU memory address for the KV matrix, the AttAcc memory address, the data size, and the direction of movement. These are repeated for all decoders.

In the Gen stage of each head, the host first offloads the QKV generation layer to xPUs. Then, the host transfers the K and V vectors, the output of the QKV generation layer, to the AttAccs by *AttAcc::MemCopy*. The Q vector is similarly transferred to the AttAccs through *AttAcc::MemCopy*, but with setting the target address to the GEMV buffer address in the AttAcc memory space. After the QKV vectors are transmitted, the host executes *AttAcc::RunAttention* to offload the attention layer to the AttAccs. *AttAcc::RunAttention* accompanies arguments including the AttAcc memory address offset of the head to be processed. During the attention layer, the host transfers the output vectors to xPUs through *AttAcc::MemCopy* with setting the direction of movement to xPUs. These are repeated for all decoders similar to the Sum stage and also repeated for the L_{out} stages. When a new stage begins, the host performs *AttAcc::UpdateRequest*, which deletes the information of the completed request with generating an end-of-sequence token, and increases L by one for the remaining requests in the config memory.

6 Maximally Utilizing AttAcc

6.1 Pipelining

Attention-level Pipelining: Within a single attention layer performed by a single HBM, the GEMV and softmax operations can be pipelined with a caveat that more than or equal to two heads are scheduled to the HBM. For instance, while the softmax operation of a head (*e.g.*, head 0) is performed on the buffer die, GEMV$_{score}$ operation for a different head (*e.g.*, head 1) can take place on the GEMV unit (see Figure 11(a)).

Head-level Pipelining: We propose a head-level pipelining of FC and attention layers to maximize the utilization of both

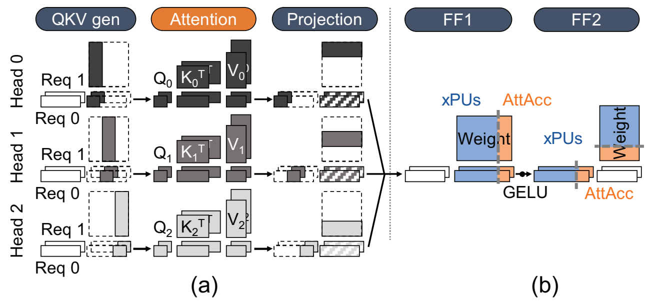

Figure 12. The concepts of (a) head-level pipelining and (b) feedforward co-processing.

xPUs and AttAccs, thus decreasing the total execution time of a single batch. Without such a pipelining, xPUs and AttAccs remain idle during the attention and FC layer, respectively, as the Gen stage or even the decoder itself operates sequentially, necessitating the previous stage's output as its input.

To avoid such an underutilization, we can exploit the two properties: i) xPUs typically exploit tiling for the FC layer due to limited on-chip cache capacity, and ii) AttAccs schedule the attention layer in a head granularity. Considering that the size of the total weight matrix easily surpasses the on-chip SRAM capacity, it is reasonable for xPUs to keep only a portion of the weight matrix. Consequently, during the QKV generation layer, only a limited number of attention head inputs will be generated in xPUs at a time, instead of the whole input generated at once (see Figure 12(a)). The same goes for the projection layer, which consumes only a portion of attention head outputs at a time. For example, for a batch size of 128 requests, an xPU can generate 128 inputs for attention heads at a time. Meanwhile, the generated attention head inputs can be immediately scheduled as AttAccs schedule the attention layer operation in a head granularity. Thus, the FC and attention layer execution time can be overlapped, increasing the utilization of both xPUs and AttAccs (see Figure 11(b)).

Batch-level Pipelining: Pipelining two or more batches at a time is indeed a possible option. With two batches in a Gen stage processed simultaneously, AttAccs can execute the attention layer of one batch (*e.g.*, batch 0) while xPUs work on the FC layer of the other batch (*e.g.*, batch 1). However, while this can effectively overlap the execution time of the FC and attention layers, such an effect can be nullified by the decreased size of the batch. To support such batch-level pipelining, the KV matrices of both batches must reside in memory. Considering that the capacity requirement of the KV matrices is already a limiting factor (§3.2), the size of each batch must be halved. This deteriorates the FC layer throughput, potentially degrading the end-to-end performance of the TbGM inference (see Figure 11(c)). Based on the quantitative analysis, we have concluded that such batch-level pipelining is more harmful than beneficial in our experimental setting.

6.2 Feedforward Co-processing

We propose to co-process the FC layers of the feedforward block on both xPUs and AttAccs. The layernorm operation between the multi-head attention block and feedforward block prevents the overlapping of the FC layer with the attention layer, even with the head-level pipelining. While the FC layers for QKV generation and projection can be pipelined with the attention layer, the FC layers of the feedforward block have to wait until all heads of the batch are complete with the attention layer. Thus, we instead choose to offload some portion of the FC layers in the feedforward block to AttAccs, which would otherwise be idle (see Figure 11(d)).

The amount of offloaded FC layers to synchronize the execution time of both xPUs and AttAccs can be calculated as follows. The GEMM performance in a batched feedforward block is bounded by off-chip memory bandwidth unless the batch size is extremely large. Thus, the ratio of the GEMM throughput for the xPUs and AttAccs is $BW_{\text{xPUs}} \times BS$: $BW_{\text{AttAccs}} \times 1$, with BS for batch size, BW_{xPUs} and BW_{AttAccs} for memory bandwidth of xPUs and AttAccs, respectively. As BW_{xPUs} and BW_{AttAccs} are constants, BS defines optimal partitioning. As the batch size can change depending on L, we place a partition of a weight matrix on xPUs assuming a maximum possible BS (minimum AttAcc offload), and place weight partitions on AttAccs assuming a minimum possible BS (maximum AttAccs offload). This results in some duplication of the weight matrix on xPUs and AttAccs, but allows for flexibility in optimal offloading for various batch sizes.

We choose to partition the weight matrix of FF1 column-wise and that of FF2 row-wise to avoid unnecessary data transfer between xPUs and AttAccs. As Figure 12(b) describes, the output of the FF1 is fed into FF2 as input, with only an element-wise GELU operation in between. Therefore, our method of partitioning allows for the output of FF1 executed in xPUs/AttAccs to stay in xPUs/AttAccs for FF2.

7 Evaluation

7.1 Experimental Setup

We compared a heterogeneous computing platform combining DGX A100 and AttAccs (*DGX+AttAccs*) with a baseline DGX-only system (DGX_{Base}) and the baseline but with twice the memory capacity (DGX_{Large}). Both DGX_{Base} and DGX_{Large} are composed of 40 HBMs, but their total memory capacity is 640GB and 1280GB, respectively, which can vary depending on the HBM configuration and stack height. *DGX* and AttAccs in *DGX+AttAccs* also consist of 40 HBMs each with a total memory capacity of 1,280GB (same as DGX_{Large}) where that of *DGX* is set to store the weight of the FC layers and that of AttAccs correspond to the rest. All HBMs used in the experiments are HBM3 with data rate of 5.2Gbps per pin. The maximum aggregate internal memory bandwidth of AttAccs is 242 TB/s, 9× the aggregate memory bandwidth of DGX_{Base} considering power constraint.

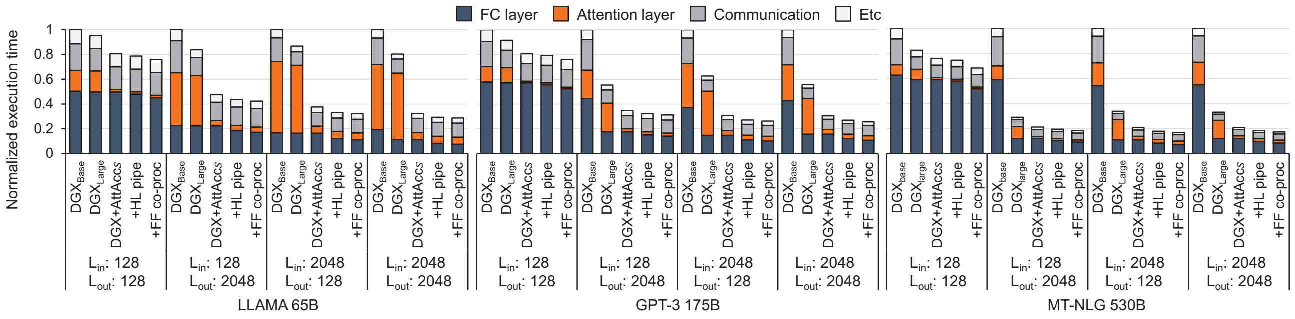

Figure 13. The normalized execution time of LLAMA 65B, GPT-3 175B, and MT-NLG 530B for various L_{in} and L_{out} to process 10,000 requests. DGX_{Base} has 640GB memory capacity and the rest have 1,280GB. +HL pipe stands for head-level pipelining applied *DGX+AttAccs*. +FF co-proc indicates that feedforward co-processing is applied to *DGX+AttAccs* in addition to +HL pipe.

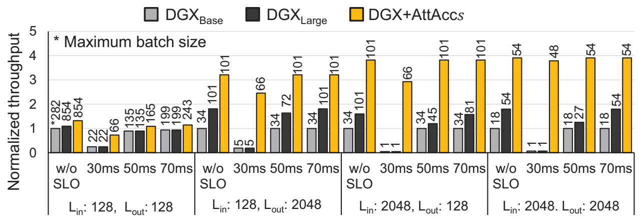

Figure 14. The normalized throughput of DGX_{Base}, DGX_{Large}, and *DGX+AttAccs* for various SLOs for GPT-3 175B. Each number above the bar represents the maximum batch size per system considering the SLO and capacity limitations.

We developed an in-house simulator by modifying Ramulator [16, 34, 41] to evaluate the performance and energy efficiency of GPU systems and AttAcc. The simulator receives system configuration and model information as input and outputs the execution time and energy consumption of each system. We validated our simulator by comparing the performance executed on the actual NVIDIA DGX A100 GPU for OPT 66B, which is known to perform similarly to the closed-source GPT-3.[3]

The energy values and area overhead of each component are calculated as follows. We designed arithmetic units in Verilog and synthesized them using Synopsys Design Compiler with a 7nm predictive process design kit (ASAP7 [10]). The GEMV unit and accumulator operate at 666MHz considering tCCDS (1.5ns) while the softmax units operate at 1.3GHz. To model SRAM-based buffers, we modified FinCACTI [55] to match the published information of SRAM [6, 24, 27, 29, 59, 64]. We scaled the area overhead of arithmetic units and buffers on the DRAM die to the third generation of 10nm class (1z-nm) DRAM process [47, 53] considering that the DRAM process is 10× less dense than the logic process of the same feature size [12]. We referred to activation and read energy of HBM from the previous work [43]. We calculated the read energy inside the DRAM die by scaling with the datapath length in HBM chip microphotographs [46, 53].

We set LLAMA 65B, GPT-3 175B, and MT-NLG 530B as the target models with the data type of FP16, FP16, and INT8, respectively. Because the model size of MT-NLG 530B is larger than the memory capacity of DGX_{Base}, we experimented with MT-NLG 530B quantized to INT8 [65]. We used the average sequence length of requests in a batch for various configurations of L_{in} and L_{out} assuming that enough requests can be batched. The SLO targets are different for each service, while publicly disclosed information is limited. Thus, we inferred the latency targets per token referring to prior works [49, 66], and explored the scenarios without the SLO, and with 30ms, 50ms, and 70ms latency targets.

7.2 Performance of AttAcc

DGX+AttAccs outperforms the conventional system by providing higher internal aggregate memory bandwidth with the PIM architecture. Figure 13 describes the normalized execution time of DGX_{Base}, DGX_{Large}, *DGX+AttAccs*, and *DGX+AttAccs* with optimizations for various models and sequence lengths to process 10,000 requests. *DGX+AttAccs* outperforms DGX_{Base} and even DGX_{Large}, which supports the same batch size as *DGX+AttAccs* in throughput, by 4.84× and 2.48×, respectively, due to significantly accelerating the attention layer by the proposed PIM architecture.

Head-level pipelining and feedforward co-processing further improve the end-to-end performance of *DGX+AttAccs*. Compared to the naïve *DGX+AttAccs*, *DGX+AttAccs* with head-level pipelining achieves an end-to-end performance improvement of up to 1.15×. Feedforward co-processing shows an additional improvement up to 1.10×. When combined, *DGX+AttAccs* with head-level pipelining and feedforward co-processing achieves performance improvement of up to 3.49× (2.81×), 3.91× (2.39×), and 5.93× (2.01×) over DGX_{Base} (DGX_{Large}) for LLAMA 65B, GPT-3 175B, and MT-NLG 530B, respectively. Hereafter, *DGX+AttAccs* refers to *DGX+AttAccs* with these two optimizations.

[3]Our simulator is located at https://github.com/scale-snu/attacc_simulator.

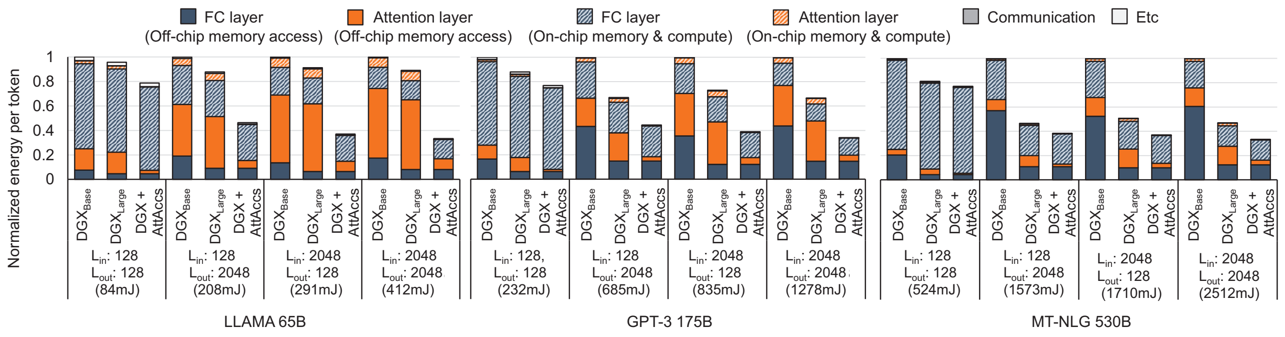

Figure 15. The normalized energy per output token of DGX_{Base}, DGX_{Large}, and *DGX+AttAccs* for LLAMA 65B, GPT-3 175B, and MT-NLG 530B without SLO. The baseline is DGX_{Base}, with absolute values in parentheses.

The performance improvement rate of *DGX+AttAccs* tends to be higher when the sequence length (L_{in}+L_{out}) is longer. Longer sequence lengths increase the size of KV matrices, reducing the maximum possible batch size. This attenuates the throughput of both FC and attention layers. In contrast, the execution time of other layers (*e.g.*, activation and normalization) and communication are rarely affected. Thus, the proportion of FC and attention layers on the total execution time of TbGM inference increases (see Figure 13). Because *DGX+AttAccs* accelerates both FC and attention layers compared to DGX_{Base}, the improvement ratio increases when the sequence length is longer.

We interpret the performance improvement of *DGX+AttAccs* over DGX_{Base} to be primarily from the acceleration of the attention layers for relatively small models (*e.g.*, LLAMA 65B) and the acceleration of the FC layers for large models (*e.g.*, MT-NLG 530B). When the model size is small, even DGX_{Base} enables large batch sizes achieving high FC layer throughput. This means that the attention layer acceleration by AttAccs becomes more noticeable. In contrast, when the model size is large, the batch size and FC layer throughput differences are more prominent. For example, the available memory capacity for KV matrices changes from 510GB in DGX_{Base} to 1,150GB in *DGX+AttAccs* for LLAMA 65B (2.3×), while it changes from 146GB in DGX_{Base} to 786GB in *DGX+AttAccs* for MT-NLG 530B (5.4×). The ratios of change in memory capacity translate to batch size and FC layer throughput.

7.3 Performance under Service-Level Objective (SLO)

When an SLO is imposed, *DGX+AttAccs* can achieve further throughput improvement than DGX_{Base} and DGX_{Large}. Figure 14 shows the throughput of DGX_{Base}, DGX_{Large}, and *DGX+AttAccs* for GPT-3 175B considering various SLOs. The batch sizes in DGX_{Base} and DGX_{Large} are limited by the SLO instead of memory capacity, especially when the SLO is tight. For example, when the SLO is 30ms for GPT-3 with L_{in}/L_{out} of 2,048, the batch size of DGX_{Large} is limited to only 1, although there is enough memory to accommodate a batch size of 48. *DGX+AttAccs* accelerates the attention layer, thereby relieving the batch size constraints caused by SLO and improving the performance. Overall, the tighter the SLO, the greater the throughput improvement of *DGX+AttAccs*. (*e.g.*, 56× than DGX_{Large} with a SLO of 30ms).

7.4 Energy Efficiency

DGX+AttAccs achieves higher energy efficiency than DGX_{Base} and DGX_{Large} (see Figure 15). First, due to the larger batch size than DGX_{Base}, the weight reusability of FC layers increases, reducing the number of off-chip memory accesses. Second, the energy consumption of the attention layer is reduced because of the memory access energy reduction of the PIM architecture, comparing to both DGX_{Base} and DGX_{Large}. As a result, the energy consumption of *DGX+AttAccs* is reduced by up to 66.7% (62.6%), 65.9% (48.8%), and 66.8% (29.1%) compared to DGX_{Base} (DGX_{Large}) for LLAMA 65B, GPT-3 175B, and MT-NLG 530B, respectively.

The longer the sequence length, the larger the energy consumption reduction of *DGX+AttAccs*. As the sequence length increases, both the attention layers and the FC layers incur more memory accesses due to larger KV matrices and smaller batch sizes, respectively. Thus, the proportion of DRAM energy consumption in total energy consumption increases, and the effect of AttAccs to reduce DRAM energy becomes more prominent. The energy reduction over DGX_{Base} is more pronounced in the attention layer for a smaller model and in the FC layer for a larger model.

7.5 Sensitivity Study of Lower Bit-Width

Even for models with reduced bit-width, AttAcc shows up to 3.47× and 2.59× performance improvement over DGX_{Base} and DGX_{Large}, respectively (see Figure 16). As the bit-width decreases, DGX_{Base} can take a larger batch size for TbGM processing. For the larger batch size, the portion of execution time for the attention layer is increased on the DGX_{Base}. The performance of AttAcc compared to DGX_{Base} decreased due to the decreased gain of increased memory capacity, but the performance of AttAcc compared to DGX_{Large} increased due to the increased gain of high aggregate memory bandwidth.

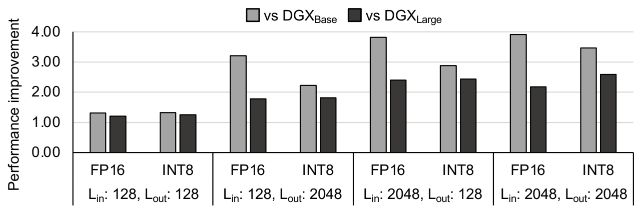

Figure 16. The performance improvement of *DGX+AttAccs* over DGX_{Base} and DGX_{Large} for GPT-3 175B when the data types of both embedding vectors and weights are half-precision floating point (FP16) or 8-bit integer (INT8).

7.6 Comparison with Other DGX Options

DGX+AttAccs outperforms the DGX system that executes the attention layer in CPUs (DGX_{CPU}) and the DGX system with twice as many GPUs as DGX_{Base} (*2×DGX*) (see Figure 17). DGX_{CPU} encompasses large CPU memory, which enables increased batch size in a single DGX system, leading to an improved FC layer throughput. However, the limited memory bandwidth of the CPU deteriorates the attention layer throughput, resulting in even a lower performance than DGX_{Base}. Scaling out DGX can be another option. *2×DGX* can achieve higher FC layer throughput than *DGX+AttAccs* because it increases the memory capacity and aggregate memory bandwidth for the FC layer. However, the aggregate bandwidth for the attention layer is 4.5× smaller than that of *DGX+AttAccs*. Even for the FC layer, the communication cost is aggravated as the number of distributed processing nodes increases. Moreover, *2×DGX* is expensive and underutilized because most of the computing resources are still idle in the attention layer. As a result, *DGX+AttAccs* shows up to 1.72× higher throughput than *2×DGX*.

7.7 Area Overhead

The total area overhead of processing units in an HBM for AttAcc is 13.12mm^2 per DRAM die and 1.40mm^2 per buffer die. The area overhead per DRAM die corresponds to 10.84% of a 121mm^2 HBM3 [46, 53].[4] Specifically, there are 16 GEMV units and 4 accumulators per pCH as shown in Figure 6 and the area overhead of each GEMV unit and accumulator implemented in a 1z-nm DRAM process is 0.094mm^2 and 0.036mm^2, respectively. In the GEMV unit, the area ratios of arithmetic units, buffers, and control logic are 63%, 14%, and 23%, respectively, while most of the accumulator's area overhead is attributed to the arithmetic units.

The control unit of the GEMV unit is simple and its area is relatively small as it only needs to process specific GEMV, unlike previous works [36, 37] that support more general operations. Conservatively assuming the area of our GEMV unit to be that of the similar PIM unit reported in previous works [36, 37], the area overhead per DRAM die is about 25%. The area overhead of the softmax unit and accumulator on the buffer die, which is implemented in a 7nm logic process, are 1.38mm^2 and 0.02mm^2, respectively. Most of the softmax unit's area overhead is attributed to the large buffers.

[4]Assuming GEMV units and accumulators that support the accumulation of full-precision floating point, the area overhead is 14.59% per DRAM die.

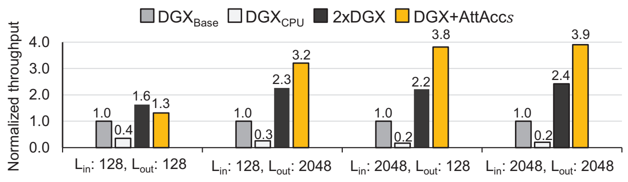

Figure 17. The normalized throughput of DGX_{Base}, DGX_{CPU}, *2×DGX*, and *DGX+AttAccs* for GPT-3 175B.

8 Discussion

Grouped/Multi-Query Attention: Multi-Query Attention (MQA) [57] shares KV matrices among all the heads, increasing the reusability and reducing the capacity requirement for the matrices. More recently introduced Grouped-Query Attention (GQA) [1] generalizes MQA and Multi-Head Attention (MHA, which is the default model and our primary target) by making a group of heads share a KV matrix pair.

AttAcc provides performance and energy efficiency gains for the attention layer by taking higher aggregate bandwidth, but as the group size of GQA increases (closer to MQA), the benefit of AttAcc diminishes. It is because, unlike AttAcc, the baseline GPU system can exploit the reusability of KV matrices with its on-chip cache and abundant compute resources. If we configure AttAcc's GEMV unit into a systolic array at a higher area cost, AttAcc can still achieve a competitive performance gain even for large group sizes by reusing the KV matrices while taking aggregate bandwidth.

Bulk Bitwise Computation: Bulk bitwise computation in DRAM [2, 14, 19, 38, 54, 69] is one of the attractive options for PIM architecture. They can provide massively parallel logic operations such as bitwise XOR/XNOR in the granularity of the DRAM row. It also incurs low area overhead due to minimal modification of DRAM microarchitecture. Although bit-serial computation would incur impractically long latency for floating-point operations, INT8 quantization applicable to TbGMs [18, 65] can be processed swiftly.

However, even when INT8 quantization is applied, bulk bitwise operations still do not outperform bank-level parallelism in the multiplication dominant attention layer. Single INT8 multiplication takes about 20us; it requires about a hundred logic operations [63], which translates to 400 AAP (activate-activate-precharge) commands [2, 54]. As a result, 8,192 multiplications per bank are performed during 20us, which can be amplified by subarray-level parallelism [7, 33]. In the case of bank-level parallelism, up to 32 INT8 multiplications can be performed every tCCDL (3ns) per bank, which corresponds to approximately 200,000 multiplications

during 20us. Moreover, bulk bitwise operations can incur higher power consumption than bank-level parallelism due to continuous row activation and subarray-level parallelism.
The Implication of AttAcc on Training: Most of the training processes of TbGM, such as pre-training, are compute-intensive, making it difficult to accelerate with AttAcc. For example, the token to be generated is already known during pre-training. Therefore, in the forward pass of pre-training, (L_{in}+L_{out}-1) tokens are processed concurrently in the attention layer with diagonal masking, making the operations compute-intensive. Meanwhile, AttAcc can be utilized in training that does not know which tokens to be generated, such as fine-tuning from human preference. Instead of learning through pre-given answers, fine-tuning from human preference generates a token and evaluates the whole sentence, including the generated token at each stage. This type of training entails memory-intensive generation stages, which are suitable for acceleration by AttAcc.

9 Related Work

Transformer Accelerators: TransPIM [69] accelerates end-to-end Transformer models with token-based data mapping and HBM-based PIM architecture. However, it targets a relatively small Transformer model (*e.g.*, GPT-2) without batching. For a large model (*e.g.*, GPT-3) with batching, end-to-end acceleration by PIM architecture is inefficient due to FC layers that require high computing power. Moreover, adopting only token-based data mapping can result in performance degradation due to the lack of the degree of parallelism of the attention layer as analyzed in §4.1. StepStone PIM [8] accelerates GEMM by enabling independent PIM execution on the CPU with address-mapping-aware GEMM blocking and unique address generation. However, StepStone does not address the TbGM inference with large batch sizes, where the GEMV of the attention layer becomes crucial.

There have been many works that have proposed accelerating the transformer model through pruning or quantization. A^3 [20] and ELSA [21] propose an algorithm that approximates the attention mechanism and an accelerator for it, focusing on the characteristic that small values among the outputs of the score operation approach near zero after the softmax. With the same motivation, SpAtten [62] and Sanger [39] proposed a method of pruning the output of score operation and proposed an accelerator for this. FACT [50] proposed eager prediction and mixed precision to optimize not only the attention layer but also the QKV generation and feedforward layers. GOBO [68] and Olive [18] proposed the quantization of weights and embedding vectors and an accelerator for this. Meanwhile, DFX [23] proposed a multi-FPGA accelerator to accelerate TbGM end-to-end losslessly and proposed a model parallelism methodology and optimized dataflow. However, these prior works for transformer acceleration mainly aimed at reducing latency and did not consider the large batch size or large model size.
Processing-In-Memory: There has been a large body of prior works executing computation closer to DRAM to exploit higher aggregate bandwidth and reduce data movement [2–4, 8, 13–15, 17, 19, 22, 25, 31, 32, 35–38, 45, 54, 58, 67]. To the best of our knowledge, AttAcc, which extended [9], is the first PIM architecture that accelerates a large TbGM with batching focusing on the attention layer. HBM-PIM [36] is the first PIM designed by a major DRAM manufacturer compatible with a JEDEC standard DRAM interface. It has reinvigorated development interests in commercial PIM devices since its introduction. Nonetheless, it needs to demonstrate its superior efficiency over conventional platforms for highly-demanding commercial applications, such as TbGM. Thus, one of the key contributions of this paper is to demonstrate the potential of such a PIM architecture for TbGM.

10 Conclusion

We have analyzed the impact of batching for transformer-based generative models and identified the growing importance of the attention layer in the trend with increasing model sizes and sequence length. The conventional serving platforms, such as GPUs, are suboptimal for large batch sizes having stringent memory capacity and bandwidth requirements in processing the attention layer under a tight service-level objective. We proposed AttAcc, an accelerator for the attention layer, composed of processing-in-memory devices exploiting the characteristic that the external bandwidth requirement is far less than the internal bandwidth requirement for the attention layer. Compared to the monolithic state-of-the-art GPU system with the same memory capacity, the heterogeneous computing platform combining GPUs with AttAccs achieved significantly higher throughput (up to 2.81×) and energy efficiency (up to 2.67×).

11 Acknowledgments

The authors thank Dr. Sukhan Lee and Dr. Byeongho Kim at Samsung for their valuable feedback on this work. This work was supported by the Samsung Electronics, under Grant IO231030-07528-01, an Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-01343 and IITP-2023-RS-2023-00256081), and a grant from PRISM, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. Nam Sung Kim has a financial interest in Samsung Electronics and NeuroRealityVision. The EDA tool was supported by the IC Design Education Center (IDEC), Korea. The ICT at Seoul National University provides research facilities for this study. Jung Ho Ahn, the corresponding author, is with the Department of Intelligence and Information and the Interdisciplinary Program in Artificial Intelligence, SNU.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints.

[2] Mustafa F. Ali, Akhilesh Jaiswal, and Kaushik Roy. 2020. In-Memory Low-Cost Bit-Serial Addition Using Commodity DRAM Technology. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 1 (2020), 155–165. https://doi.org/10.1109/TCSI.2019.2945617

[3] Bahar Asgari, Ramyad Hadidi, Jiashen Cao, Da Eun Shim, Sung-Kyu Lim, and Hyesoon Kim. 2021. FAFNIR: Accelerating Sparse Gathering by Using Efficient Near-Memory Intelligent Reduction. In *HPCA*. 908–920. https://doi.org/10.1109/HPCA51647.2021.00080

[4] Hadi Asghari-Moghaddam, Young Hoon Son, Jung Ho Ahn, and Nam Sung Kim. 2016. Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems. In *MICRO*. 1–13. https://doi.org/10.1109/MICRO.2016.7783753

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *NeurIPS*. https://doi.org/10.48550/arXiv.2005.14165

[6] Jonathan Chang, Yen-Huei Chen, Wei-Min Chan, Sahil Preet Singh, Hank Cheng, Hidehiro Fujiwara, Jih-Yu Lin, Kao-Cheng Lin, John Hung, Robin Lee, Hung-Jen Liao, Jhon-Jhy Liaw, Quincy Li, Chih-Yung Lin, Mu-Chi Chiang, and Shien-Yang Wu. 2017. A 7nm 256Mb SRAM in High-K Metal-Gate FinFET Technology with Write-Assist Circuitry for Low-VMIN Applications. In *IEEE International Solid-State Circuits Conference*. 206–207. https://doi.org/10.1109/ISSCC.2017.7870333

[7] Kevin K Chang, Prashant J Nair, Donghyuk Lee, Saugata Ghose, Moinuddin K Qureshi, and Onur Mutlu. 2016. Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Movement in DRAM. In *HPCA*. https://doi.org/10.1109/HPCA.2016.7446095

[8] Benjamin Y. Cho, Jeageun Jung, and Mattan Erez. 2021. Accelerating Bandwidth-Bound Deep Learning Inference with Main-Memory Accelerators. In *SC21: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14. https://doi.org/10.1145/3458817.3476146

[9] Jaewan Choi, Jaehyun Park, Kwanhee Kyung, Nam Sung Kim, and Jung Ho Ahn. 2023. Unleashing the Potential of PIM: Accelerating Large Batched Inference of Transformer-Based Generative Models. *IEEE Computer Architecture Letters* 22, 02 (2023), 113–116. https://doi.org/10.1109/LCA.2023.3305386

[10] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. 2016. ASAP7: A 7-nm FinFET Predictive Process Design Kit. *Microelectronics Journal* 53 (2016), 105–115.

[11] Compute Express Link Consortium. 2022. Compute Express Link 3.0 White Paper. https://www.computeexpresslink.org/_files/ugd/0c1418_a8713008916044ae9604405d10a7773b.pdf

[12] Fabrice Devaux. 2019. The true Processing In Memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*. 1–24. https://doi.org/10.1109/HOTCHIPS.2019.8875680

[13] Amin Farmahini-Farahani, Jung Ho Ahn, Katherine Morrow, and Nam Sung Kim. 2015. NDA: Near-DRAM Acceleration Architecture Leveraging Commodity DRAM Devices and Standard Memory Modules. In *HPCA*. 283–295. https://doi.org/10.1109/HPCA.2015.7056040

[14] Fei Gao, Georgios Tziantzioulis, and David Wentzlaff. 2019. ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs. In *MICRO*. 100–113. https://doi.org/10.1145/3352460.3358260

[15] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. 2017. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. In *ASPLOS*. 751–764. https://doi.org/10.1145/3037697.3037702

[16] SAFARI Research Group. 2023. Ramulator 2.0 — GitHub Repository. https://github.com/CMU-SAFARI/ramulator2

[17] Peng Gu, Xinfeng Xie, Yufei Ding, Guoyang Chen, Weifeng Zhang, Dimin Niu, and Yuan Xie. 2020. iPIM: Programmable In-Memory Image Processing Accelerator Using Near-Bank Architecture. In *ISCA*. 804–817. https://doi.org/10.1109/ISCA45697.2020.00071

[18] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyi Guo, and Yuhao Zhu. 2023. OliVe: Accelerating Large Language Models via Hardware-Friendly Outlier-Victim Pair Quantization. In *ISCA*. https://doi.org/10.1145/3579371.3589038

[19] Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, João Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gómez-Luna, and Onur Mutlu. 2021. SIMDRAM: A Framework for Bit-Serial SIMD Processing Using DRAM. In *ASPLOS*. 329–345. https://doi.org/10.1145/3445814.3446749

[20] Tae Jun Ham, Sung Jun Jung, Seonghak Kim, Young H. Oh, Yeonhong Park, Yoonho Song, Jung-Hun Park, Sanghee Lee, Kyoung Park, Jae W. Lee, and Deog-Kyoon Jeong. 2020. A^3: Accelerating Attention Mechanisms in Neural Networks with Approximation. In *HPCA*. 328–341. https://doi.org/10.1109/HPCA47549.2020.00035

[21] Tae Jun Ham, Yejin Lee, Seong Hoon Seo, Soosung Kim, Hyunji Choi, Sung Jun Jung, and Jae W. Lee. 2021. ELSA: Hardware-Software Codesign for Efficient, Lightweight Self-Attention Mechanism in Neural Networks. In *ISCA*. 692–705. https://doi.org/10.1109/ISCA52012.2021.00060

[22] Mingxuan He, Choungki Song, Ilkon Kim, Chunseok Jeong, Seho Kim, Il Park, Mithuna Thottethodi, and T. N. Vijaykumar. 2020. Newton: A DRAM-maker's Accelerator-in-Memory (AiM) Architecture for Machine Learning. In *MICRO*. 372–385. https://doi.org/10.1109/MICRO50266.2020.00040

[23] Seongmin Hong, Seungjae Moon, Junsoo Kim, Sungjae Lee, Minsub Kim, Dongsoo Lee, and Joo-Young Kim. 2022. DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation. In *MICRO*. https://doi.org/10.1109/MICRO56248.2022.00051

[24] IEEE. 2018. *International Roadmap for Devices and Systems: 2018.* Technical Report. https://irds.ieee.org/editions/2018/

[25] Mohsen Imani, Saransh Gupta, Yeseong Kim, and Tajana Rosing. 2019. FloatPIM: In-Memory Acceleration of Deep Neural Network Training with High Precision. In *ISCA*. 802–815. https://doi.org/10.1145/3307650.3322237

[26] JEDEC. 2022. High Bandwidth Memory DRAM (HBM3).

[27] W.C. Jeong, S. Maeda, H.J. Lee, K.W. Lee, T.J. Lee, D.W. Park, B.S. Kim, J.H. Do, T. Fukai, D.J. Kwon, K.J. Nam, W.J. Rim, M.S. Jang, H.T. Kim, Y.W. Lee, J.S. Park, E.C. Lee, D.W. Ha, C.H. Park, H.J. Cho, S.M. Jung, and H.K. Kang. 2018. True 7nm Platform Technology featuring Smallest FinFET and Smallest SRAM cell by EUV, Special Constructs and 3rd Generation Single Diffusion Break. In *IEEE Symposium on VLSI Technology*. 59–60. https://doi.org/10.1109/VLSIT.2018.8510682

[28] Norm Jouppi, George Kurian, Sheng Li, Peter Ma, Rahul Nagarajan, Lifeng Nai, Nishant Patil, Suvinay Subramanian, Andy Swing, Brian Towles, Clifford Young, Xiang Zhou, Zongwei Zhou, and David A Patterson. 2023. TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings. In *ISCA*. https://doi.org/10.1145/3579371.3589350

[29] Norman P. Jouppi, Doe Hyun Yoon, Matthew Ashcraft, Mark Gottscho, Thomas B. Jablin, George Kurian, James Laudon, Sheng Li, Peter C. Ma, Xiaoyu Ma, Thomas Norrie, Nishant Patil, Sushma Prasad, Cliff Young, Zongwei Zhou, and David A. Patterson. 2021. Ten Lessons From Three Generations Shaped Google's TPUv4i: Industrial Product. In *ISCA*. 1–14. https://doi.org/10.1109/ISCA52012.2021.00010

[30] Hongshin Jun, Jinhee Cho, Kangseol Lee, Ho-Young Son, Kwiwook Kim, Hanho Jin, and Keith Kim. 2017. HBM (High Bandwidth Memory)

DRAM Technology and Architecture. In *2017 IEEE International Memory Workshop (IMW)*. 1–4. https://doi.org/10.1109/IMW.2017.7939084

[31] Byeongho Kim, Jongwook Chung, Eojin Lee, Wonkyung Jung, Sunjung Lee, Jaewan Choi, Jaehyun Park, Minbok Wi, Sukhan Lee, and Jung Ho Ahn. 2020. MViD: Sparse Matrix-Vector Multiplication in Mobile DRAM for Accelerating Recurrent Neural Networks. *IEEE Trans. Comput.* 69, 7 (2020), 955–967. https://doi.org/10.1109/TC.2020.2984496

[32] Duckhwan Kim, Jaeha Kung, Sek Chai, Sudhakar Yalamanchili, and Saibal Mukhopadhyay. 2016. Neurocube: A Programmable Digital Neuromorphic Architecture with High-Density 3D Memory. In *ISCA*. 380–392. https://doi.org/10.1109/ISCA.2016.41

[33] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. 2012. A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM. In *ISCA*. https://doi.org/10.1145/2366231.2337202

[34] Yoongu Kim, Weikun Yang, and Onur Mutlu. 2016. Ramulator: A Fast and Extensible DRAM Simulator. *IEEE Computer Architecture Letters* 15, 1 (2016), 45–49. https://doi.org/10.1109/LCA.2015.2414456

[35] Youngeun Kwon, Yunjae Lee, and Minsoo Rhu. 2019. TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning. In *MICRO*. 740–753. https://doi.org/10.1145/3352460.3358284

[36] Sukhan Lee, Shin-haeng Kang, Jaehoon Lee, Hyeonsu Kim, Eojin Lee, Seungwoo Seo, Hosang Yoon, Seungwon Lee, Kyounghwan Lim, Hyunsung Shin, Jinhyun Kim, O Seongil, Anand Iyer, David Wang, Kyomin Sohn, and Nam Sung Kim. 2021. Hardware Architecture and Software Stack for PIM based on Commercial DRAM Technology: Industrial Product. In *ISCA*. https://doi.org/10.1109/ISCA52012.2021.00013

[37] Seongju Lee, Kyuyoung Kim, Sanghoon Oh, Joonhong Park, Gimoon Hong, Dongyoon Ka, Kyudong Hwang, Jeongje Park, Kyeongpil Kang, Jungyeon Kim, Junyeol Jeon, Nahsung Kim, Yongkee Kwon, Kornijcuk Vladimir, Woojae Shin, Jongsoon Won, Minkyu Lee, Hyunha Joo, Haerang Choi, Jaewook Lee, Donguc Ko, Younggun Jun, Keewon Cho, Ilwoong Kim, Choungki Song, Chunseok Jeong, Daehan Kwon, Jieun Jang, Il Park, Junhyun Chun, and Joohwan Cho. 2022. A 1ynm 1.25V 8Gb, 16Gb/s/pin GDDR6-based Accelerator-in-Memory supporting 1TFLOPS MAC Operation and Various Activation Functions for Deep-Learning Applications. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 1–3.

[38] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: A DRAM-Based Reconfigurable In-Situ Accelerator. In *MICRO*. 288–301. https://doi.org/10.1145/3123939.3123977

[39] Liqiang Lu, Yicheng Jin, Hangrui Bi, Zizhang Luo, Peng Li, Tao Wang, and Yun Liang. 2021. Sanger: A Co-Design Framework for Enabling Sparse Attention using Reconfigurable Architecture. In *MICRO*. https://doi.org/10.1145/3466752.3480125

[40] David Luebke. 2008. CUDA: Scalable Parallel Programming for High-Performance Scientific Computing. In *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro (ISBI)*. 836–838. https://doi.org/10.1109/ISBI.2008.4541126

[41] Haocong Luo, Yahya Can Tuğrul, F. Nisa Bostancı, Ataberk Olgun, A. Giray Yağlıkçı, , and Onur Mutlu. 2023. Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator.

[42] NVIDIA. 2023. NVIDIA DGX A100. https://resources.nvidia.com/en-us-dgx-systems/dgx-ai

[43] Mike O'Connor, Niladrish Chatterjee, Donghyuk Lee, John Wilson, Aditya Agrawal, Stephen W Keckler, and William J Dally. 2017. Fine-Grained DRAM: Energy-Efficient DRAM for Extreme Bandwidth Systems. In *MICRO*. https://doi.org/10.1145/3123939.3124545

[44] OpenAI. 2023. Models. https://platform.openai.com/docs/models/.

[45] Jaehyun Park, Byeongho Kim, Sungmin Yun, Eojin Lee, Minsoo Rhu, and Jung Ho Ahn. 2021. TRiM: Enhancing Processor-Memory Interfaces with Scalable Tensor Reduction in Memory. In *MICRO*. https://doi.org/10.1145/3466752.3480080

[46] Myeong-Jae Park, Ho Sung Cho, Tae-Sik Yun, Sangjin Byeon, Young Jun Koo, Sangsic Yoon, Dong Uk Lee, Seokwoo Choi, Jihwan Park, Jinhyung Lee, Kyungjun Cho, Junil Moon, Byung-Kuk Yoon, Young-Jun Park, Sang-muk Oh, Chang Kwon Lee, Tae-Kyun Kim, Seong-Hee Lee, Hyun-Woo Kim, Yucheon Ju, Seung-Kyun Lim, Seung Geun Baek, Kyo Yun Lee, Sang Hun Lee, Woo Sung We, Seungchan Kim, Yongseok Choi, Seong-Hak Lee, Seung Min Yang, Gunho Lee, In-Keun Kim, Younghyun Jeon, Jae-Hyung Park, Jong Chan Yun, Chanhee Park, Sun-Yeol Kim, Sungjin Kim, Dong-Yeol Lee, Su-Hyun Oh, Taejin Hwang, Junghyun Shin, Yunho Lee, Hyunsik Kim, Jaeseung Lee, Youngdo Hur, Sangkwon Lee, Jieun Jang, Junhyun Chun, and Joohwan Cho. 2022. A 192-Gb 12-High 896-GB/s HBM3 DRAM with a TSV Auto-Calibration Scheme and Machine-Learning-Based Layout Optimization. In *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 65. 444–446. https://doi.org/10.1109/ISSCC42614.2022.9731562

[47] Myeong-Jae Park, Jinhyung Lee, Kyungjun Cho, Jihwan Park, Junil Moon, Sung-Hak Lee, Tae-Kyun Kim, Sanghoon Oh, Seokwoo Choi, Yongsuk Choi, Ho Sung Cho, Taesik Yun, Young Jun Koo, Jae-Seung Lee, Byung-Kuk Yoon, Young-Jun Park, Sangmuk Oh, Chang Kwon Lee, Seong-Hee Lee, Hyun-Woo Kim, Yucheon Ju, Seung-Kyun Lim, Kyo Yun Lee, Sang-Hoon Lee, Woo Sung We, Seungchan Kim, Seung Min Yang, Keonho Lee, In-Keun Kim, Younghyun Jeon, Jae-Hyung Park, Jong Chan Yun, Seonyeol Kim, Dong-Yeol Lee, Su-Hyun Oh, Jung-Hyun Shin, Yeonho Lee, Jieun Jang, and Joohwan Cho. 2023. A 192-Gb 12-High 896-GB/s HBM3 DRAM With a TSV Auto-Calibration Scheme and Machine-Learning-Based Layout Optimization. *IEEE Journal of Solid-State Circuits* 58, 1 (2023), 256–269. https://doi.org/10.1109/JSSC.2022.3193354

[48] Naebeom Park, Sungju Ryu, Jaeha Kung, and Jae-Joon Kim. 2021. High-throughput near-memory processing on CNNs with 3D HBM-like memory. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 6 (2021), 1–20. https://doi.org/10.1145/3460971

[49] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Aashaka Shah, Saeed Maleki, and Ricardo Bianchini. 2023. Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677 [cs.AR]

[50] Yubin Qin, Yang Wang, Dazheng Deng, Zhiren Zhao, Xiaolong Yang, Leibo Liu, Shaojun Wei, Yang Hu, and Shouyi Yin. 2023. FACT: FFN-Attention Co-Optimized Transformer Architecture with Eager Correlation Prediction. In *ISCA*. https://doi.org/10.1145/3579371.3589057

[51] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving Language Understanding by Generative Pre-training. (2018).

[52] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[53] Yesin Ryu, Sung-Gi Ahn, Jae Hoon Lee, Jaewon Park, Yong Ki Kim, Hyochang Kim, Yeong Geol Song, Han-Won Cho, Sunghye Cho, Seung Ho Song, Haesuk Lee, Useung Shin, Jonghyun Ahn, Je-Min Ryu, Sukhan Lee, Kyoung-Hwan Lim, Jungyu Lee, Jeong Hoan Park, Jae-Seung Jeong, Sunghwan Joo, Dajung Cho, So Young Kim, Minsu Lee, Hyunho Kim, Minhwan Kim, Jae-San Kim, Jinah Kim, Hyun Gil Kang, Myung-Kyu Lee, Sung-Rae Kim, Young-Cheon Kwon, Young Yong Byun, Kijun Lee, Sangkil Park, Jaeyoun Youn, Myeong-O Kim, Kyomin Sohn, Sang-Joon Hwang, and Jooyoung Lee. 2023. A 16 GB 1024 GB/s HBM3 DRAM With Source-Synchronized Bus Design and On-Die Error Control Scheme for Enhanced RAS Features. *IEEE Journal of Solid-State Circuits* 58, 4 (2023), 1051–1061. https://doi.org/10.1109/JSSC.2022.3232096

[54] Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry. 2017. Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology. In *MICRO*. 273–287. https://doi.org/10.1145/3123939.3124544

[55] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. 2014. FinCACTI: Architectural Analysis and Modeling of Caches with Deeply-Scaled FinFET Devices. In *2014 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 290–295. https://doi.org/10.1109/ISVLSI.2014.94

[56] Debendra Das Sharma. 2020. PCI Express® 6.0 Specification at 64.0 GT/s with PAM-4 signaling: a low latency, high bandwidth, high reliability and cost-effective interconnect. In *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*. 1–8. https://doi.org/10.1109/HOTI51249.2020.00016

[57] Noam Shazeer. 2019. Fast Transformer Decoding: One Write-Head is All You Need.

[58] Hyunsung Shin, Dongyoung Kim, Eunhyeok Park, Sungho Park, Yongsik Park, and Sungjoo Yoo. 2018. McDRAM: Low Latency and Energy-Efficient Matrix Computations in DRAM. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2613–2622. https://doi.org/10.1109/TCAD.2018.2857044

[59] Taejoong Song, Jonghoon Jung, Woojin Rim, Hoonki Kim, Yongho Kim, Changnam Park, Jeongho Do, Sunghyun Park, Sungwee Cho, Hyuntaek Jung, Bongjae Kwon, Hyun-Su Choi, Jaeseung Choi, and Jong Shik Yoon. 2018. A 7nm FinFET SRAM Using EUV Lithography with Dual Write-Driver-Assist Circuitry for Low-Voltage Applications. In *IEEE International Solid-State Circuits Conference*. 198–200. https://doi.org/10.1109/ISSCC.2018.8310252

[60] John E Stone, David Gohara, and Guochun Shi. 2010. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering* 12, 3 (2010), 66. https://doi.org/10.1109/MCSE.2010.69

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. https://doi.org/10.48550/arXiv.1706.03762

[62] Hanrui Wang, Zhekai Zhang, and Song Han. 2021. SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning. In *HPCA*. 97–110. https://doi.org/10.1109/HPCA51647.2021.00018

[63] Jingcheng Wang, Xiaowei Wang, Charles Eckert, Arun Subramaniyan, Reetuparna Das, David Blaauw, and Dennis Sylvester. 2020. A 28-nm Compute SRAM With Bit-Serial Logic/Arithmetic Operations for Programmable In-Memory Vector Computing. *IEEE Journal of Solid-State Circuits* 55, 1 (2020), 76–86. https://doi.org/10.1109/JSSC.2019.2939682

[64] Shien-Yang Wu, C.Y. Lin, M.C. Chiang, J.J. Liaw, J.Y. Cheng, S.H. Yang, C.H. Tsai, P.N. Chen, T. Miyashita, C.H. Chang, V.S. Chang, K.H. Pan, J.H. Chen, Y.S. Mor, K.T. Lai, C.S. Liang, H.F. Chen, S.Y. Chang, C.J. Lin, C.H. Hsieh, R.F. Tsui, C.H. Yao, C.C. Chen, R. Chen, C.H. Lee, H.J. Lin, C.W. Chang, K.W. Chen, M.H. Tsai, K.S. Chen, Y. Ku, and S.M. Jang. 2016. A 7nm CMOS Platform Technology Featuring 4th Generation FinFET Transistors with a 0.027um2 High Density 6-T SRAM cell for Mobile SoC Applications. In *IEEE International Electron Devices Meeting*. 2.6.1–2.6.4. https://doi.org/10.1109/IEDM.2016.7838333

[65] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. In *Proceedings of the 40th International Conference on Machine Learning*, Vol. 202. 38087–38099. https://proceedings.mlr.press/v202/xiao23c.html

[66] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. ORCA: A Distributed Serving System for Transformer-Based Generative Models. In *OSDI*. https://www.usenix.org/conference/osdi22/presentation/yu

[67] Sungmin Yun, Byeongho Kim, Jaehyun Park, Hwayong Nam, Jung Ho Ahn, and Eojin Lee. 2022. GraNDe: Near-Data Processing Architecture With Adaptive Matrix Mapping for Graph Convolutional Networks. *IEEE Computer Architecture Letters* 21, 2 (2022), 45–48. https://doi.org/10.1109/LCA.2022.3182387

[68] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *MICRO*. 811–824. https://doi.org/10.1109/MICRO50266.2020.00071

[69] Minxuan Zhou, Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. TransPIM: A Memory-based Acceleration via Software-Hardware Co-Design for Transformer. In *HPCA*. https://doi.org/10.1109/HPCA53966.2022.00082