Cambricon-LLM: A Chiplet-Based Hybrid Architecture for On-Device Inference of 70B LLM

Zhongkai Yu[1,2,†], Shengwen Liang[1,†], Tianyun Ma[3], Yunke Cai[1,2], Ziyuan Nan[1,2], Di Huang[1], Xinkai Song[1], Yifan Hao[1], Jie Zhang[4], Tian Zhi[1], Yongwei Zhao[1], Zidong Du[1,5], Xing Hu[1,5,*], Qi Guo[1], Tianshi Chen[6]

[1]SKL of Processors, Institute of Computing Technology, CAS, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]University of Science and Technology of China, Beijing, China
[4]Peking University, Beijing, China
[5]Shanghai Innovation Center for Processor Technologies
[6]Cambricon Technologies Co., Ltd., China
Emails: {yuzhongkai21s, liangshengwen, caiyunke21e, nanziyuan21s, huangdi, songxinkai, haoyifan, zhitian, zhaoyongwei, duzidong, huxing, guoqi}@ict.ac.cn, mty21@mail.ustc.edu.cn, jiez@pku.edu.cn, tchen@cambricon.com

Abstract—**Deploying advanced large language models on edge devices, such as smartphones and robotics, is a growing trend that enhances user data privacy and network connectivity resilience while preserving intelligent capabilities. However, such a task exhibits single-batch computing with incredibly low arithmetic intensity, which poses the significant challenges of huge memory footprint and bandwidth demands on limited edge resources.**

To address these issues, we introduce Cambricon-LLM, a chiplet-based hybrid architecture with NPU and a dedicated NAND flash chip to enable efficient on-device inference of 70B LLMs. Such a hybrid architecture utilizes both the high computing capability of NPU and the data capacity of the NAND flash chip, with the proposed hardware-tiling strategy that minimizes the data movement overhead between NPU and NAND flash chip. Specifically, the NAND flash chip, enhanced by our innovative in-flash computing and on-die ECC techniques, excels at performing precise lightweight on-die processing. Simultaneously, the NPU collaborates with the flash chip for matrix operations and handles special function computations beyond the flash's on-die processing capabilities. Overall, Cambricon-LLM enables the on-device inference of 70B LLMs at a speed of 3.44 token/s, and 7B LLMs at a speed of 36.34 token/s, which is over 22× to 45× faster than existing flash-offloading technologies, showing the potentiality of deploying powerful LLMs in edge devices.

Index Terms—**In-Flash Computing; Large Language Model Accelerator; Robotic Accelerator**

I. Introduction

Large Language Models (LLMs) have demonstrated remarkable performance across a variety of tasks, suggesting their potential to reshape labor markets and boost human productivity. As such potential unfolds, it is increasingly important to facilitate private LLM inferences on edge devices, such as robotics or even smartphones. By ensuring that everyone possesses a personalized LLM agent, we not only harness the transformative power of LLM agents but also cater to fundamental needs of privacy, customization, and accessibility.

However, the deployment of personal LLM agents on edge devices, where the predominant task type is single-batch

---

*Corresponding author
†Equal contribution

computing, presents unique challenges compared to traditional LLM inference processes in the cloud. This mode of inference does not leverage inter-batch parallelism, exacerbating memory bandwidth bottlenecks and further restricting the deployment of LLMs on edge devices. Specifically, personal LLM deployment faced the following challenges:

● **Huge memory footprint on limited edge resources.** Research has consistently demonstrated that LLMs with a greater number of parameters exhibit enhanced emergent capabilities and superior performance [78]. However, the extensive parameter size presents considerable challenges in terms of memory, as exemplified by the Llama-70B model under INT8 quantization, which demands 70GB of memory. This requirement far exceeds the capacity of typical smartphone DRAMs. Furthermore, large parameter sizes inevitably lead to intensive data movement, which is the primary source of energy consumption during LLM single-batch inference on edge devices. Notably, the energy cost associated with moving a single bit of data is estimated to be 100-500× greater than that required for computation [21], [48], [51], [56].

● **Substantial bandwidth demand due to single-batch's incredibly low arithmetic intensity.** Arithmetic Intensity is defined as the ratio of the computational operations to the amount of data transferred between slow and fast memory. A Low arithmetic intensity suggests that the program is memory-bound. Unfortunately, single-batch LLM inference holds an unprecedentedly low arithmetic intensity of 2 under INT8 quantization. As shown in Figure 1(a), the arithmetic intensity of LLM single-batch inference is 30× to 100× lower than that of other AI algorithms like DLRM [49], BERT [14] and VGG [61], and over 100× lower than the capabilities of hardware, such as Apple A16, NVIDIA A100 and NVIDIA Jetson Orin. This gap leads to extremely low hardware utilization due to the pronounced memory access bottlenecks and an extremely high demand for bandwidth.

To address the issue of the huge memory footprint, several works such as FlexGen [60] and DeepSpeed [54] have proposed offloading LLMs to flash-based SSDs. However, this
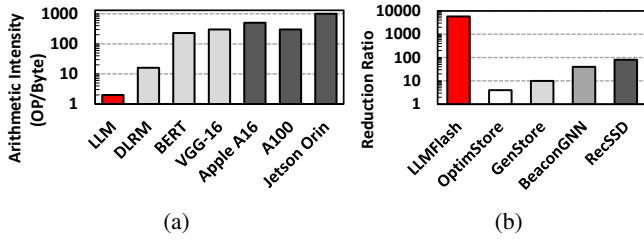
Fig. 1: (a) Arithmetic Intensity comparison between LLM and other AI algorithms and (b) Reduction Ratio comparison between the scenario in LLMFlash and other ISC works.

approach has notable limitations. Firstly, the limited bandwidth of flash poses a critical bottleneck. For example, offloading the inference of INT8 quantized Llama2-70B model on UFS 4.0 flash (with 4GB/s bandwidth) yields a theoretical maximum speed of only 0.06 token/s, far from sufficient for real-time interactive applications that require a minimum of 3-10 token/s [5], [25], [27], [50]. Secondly, extra data transfer challenges the energy budget. Conventional architectures prevent NPUs from directly accessing data on flash, necessitating an initial data migration to DRAM, increasing the total data transfer by over 3×. Given that data movement energy dominates the total energy consumption, these extra data transfers can result in significant energy overhead. To meet the huge bandwidth demands, in-storage computing (ISC) seems to be a potential solution by exploiting the inherent parallelism of flash. However, even the latest ISC works featuring on-die processing struggle to accommodate our scenario (*e.g.* OptimStore [31] and BeaconGNN [68]). Firstly, the high reduction ratio of LLM single-batch inference results in suboptimal flash channel utilization of under 10%. The reduction ratio is defined as the ratio of input data size to output data size for a given operator. Taking the Llama2-7B model as an example, where over 95% of computations involve general matrix-vector (GeMV) multiplications and the smallest matrix size is 4096×4096. After the computation, the result vector is reduced in size by a factor of 4096 compared to the original weight matrices. As shown in Figure 1(b), the reduction ratio of LLM single-batch inference is 100× larger than that observed in other scenarios discussed in previous ISC research, making them unsuitable. Secondly, the absence of an on-die error correction mechanism in OptimStore and BeaconGNN leads to unreliable outcomes. If LLMs are exposed to flash memory errors without protection, their accuracy can decrease dramatically by over 70%.

To this end, we propose Cambricon-LLM, a chiplet-based hybrid architecture with NPU and an on-die processing flash, to enable the efficient inference of 70B-LLMs on edge devices. As shown in Figure 4(a), Cambricon-LLM features a dedicated flash chip directly connected to the NPU via a chiplet die-to-die (D2D) link. This design effectively utilizes the data capacity of the flash chip and the computing capability of NPU, with the optimized tiling strategy that minimizes the data transfer between them. The integration of chiplet technology not only frees up flash from the bandwidth limitations of the UFS 4.0

protocol but also facilitates low-energy data transfers between NPU and flash, with an integrated flash controller for direct access from NPU to flash data. Furthermore, the flash memory within Cambricon-LLM is equipped with on-die computation capabilities, allowing it to fully explore the inner bandwidth and reduce the data traffic on the flash channels. In observing the reliability issue of in-storage computing architecture due to the frequent flash errors, Cambricon-LLM proposes an ultra-lightweight on-die Error Correction Unit to protect the outliers of model weights, thereby ensuring the accuracy of LLM inference.

To the best of our knowledge, Cambricon-LLM is the first hybrid architecture to extend NPU with a dedicated flash to achieve efficient single-batch inference of LLMs on edge devices. We summarize the contributions of our work as follows:

- We propose Cambricon-LLM, a novel chiplet-based hybrid architecture that combines an NPU and a specially designed flash. The flash is equipped with on-die processing capabilities and tailored for edge-based LLM inference.
- We introduce a hardware-aware tiling strategy that optimally distributes the LLM inference workload between NPU and flash, therefore fully using the available resources and avoiding flash channel underutilization.
- We propose an efficient on-die error correction algorithm and a lightweight hardware implementation, which effectively neutralizes the high error rates of flash memory, preserving the accuracy of LLM inference.
- We employ the SSDsim simulator [26] to evaluate Cambricon-LLM under various channel and chip configurations. Cambricon-LLM achieves an inference speed of 3.44 tokens/s for 70B LLM, which is over 22× faster than the baseline.

II. Background

A. LLM Inference

State-of-the-art LLMs [2], [4], [9], [16], [65], [78] are composed of multi-layered architectures where each layer corresponds to the decoder part of a Transformer model [66]. These LLMs feature an autoregressive computation pattern, where each output token is used as the input for generating the next token, leading to a sequential token generation process. To minimize redundant computations of this process, the key and value vectors of previously generated tokens are stored for use in subsequent token computations, referred to as the KV cache. The size of the KV cache is proportionally related to both the batch size and the sequence length. In edge-based LLM inference, which typically uses a batch size of one, the KV cache remains small. For example, a 70B parameter LLM with a sequence length of 1000 would require a KV cache of around 700MB, a manageable size for the DRAM of edge devices, thus making on-device inference possible.

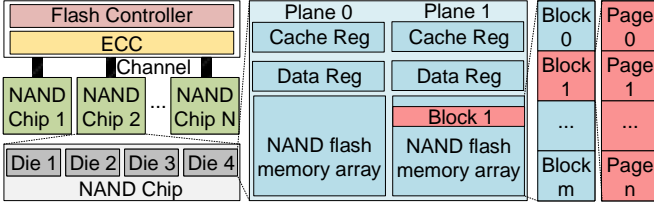The inference process of Large Language Models (LLM) is divided into two distinct phases:

Fig. 2: Architecture of Flash Memory.

The prefill phase generates the first new token, x_{m+1}, based on the m user-input tokens. Concurrently, the key and value vectors for the initial m tokens are computed and stored in DRAM as KV cache. Since all initial token values are known, these computations can occur in parallel, minimizing memory bottlenecks and allowing for efficient processing by existing NPU architectures.

The decode phase involves sequentially generating new tokens. For instance, to generate the $(t + 1)^{\text{th}}$ token, the model takes the tth token as input and processes it through the layers to output the $(t + 1)^{\text{th}}$ token. Notably, the keys and values for all previous tokens have already been computed and stored, thus avoiding redundant calculations. Although the generation of each new token has extremely low computational demand, it necessitates a whole transfer of weights to the chip, presenting a significant memory bottleneck and posing the greatest challenge for LLM inference on edge devices. For example, for Llama-70B with INT8 quantization, the generation of a token requires only around 0.14 Tera operations but more than 70GB of memory access.

B. In-storage Computing Based on Flash

Flash memory is a high-density, non-volatile storage medium that provides smartphones with over 1 TB of storage capacity. Figure 2 illustrates the primary architecture of flash memory. Within the flash, there are multiple independent channels, each externally connected to a corresponding Error-Correcting Code (ECC) module and flash controller. Each independent channel is equipped with several chips, sharing the bandwidth of the channel. In other words, only one chip can communicate with the channel at any given moment. The traditional hierarchy of flash memory includes channels, chips, dies, planes, blocks, and pages. The size of each page is 4, 8, or 16 KB, with each plane containing two registers of equivalent size to a page for data buffering. Read operations in flash memory are performed at the page level, whereas write operations are conducted at the block level. Write operations require a duration that is one to two orders of magnitude longer than read operations.

To break the bandwidth bottleneck of flash, numerous studies have moved computational modules to storage devices, known as in-storage computing (ISC). Currently, there are two main types of ISC technologies based on flash memory: on-controller computing [39], [45], [69], [71] and on-die computing [31], [44], [68]. **On-controller computing** integrates specialized architectures and hardware resources at the flash controller side, allowing it to process computational tasks directly, and fully utilizing the bandwidth of all parallel flash channels. This design, commonly used in SSDs, helps overcome the limitations of external PCIe interface bandwidth. For instance, while PCIe 4.0 SSDs have an interface bandwidth of only 8 GB/s, the total internal channel bandwidth can exceed 20 GB/s, making on-controller computing an effective solution. However, this design does not address the issue of insufficient bandwidth within individual channels.

On-die computing refers to the addition of computational units within the die to reduce the volume of data transfer across channels. This design maximally exposes the parallel bandwidth of different chips/dies/planes to the computational units, thereby alleviating the pressure of data transmission on the channels. However, the limited space within the die restricts the complexity of computational logic that can be implemented. Additionally, moving computational units inside the die can render external ECC correction modules ineffective, potentially leading to computational errors.

III. MOTIVATION

In this section, we present the motivation of Cambricon-LLM architecture. Section III-A emphasizes the unique characteristics of LLM inference tasks on edge devices. Section III-B justifies why flash memory is an ideal storage medium for this task. In Section III-C, we conduct experiments to highlight the necessity of error-correction functionality for implementing high-accuracy LLM inference using flash memory.

A. Low Arithmetic Intensity of Single-batch LLM Inference

Arithmetic intensity quantifies the ratio of an algorithm's computational workload to its memory access volume. A lower arithmetic intensity suggests a more pronounced memory-bound condition. Remarkably, the decode phase of single-batch LLM inference under INT8 quantization shows an unprecedentedly low arithmetic intensity of 2. This can be attributed to two primary reasons: Firstly, the decoder-only architecture prevents prominent LLMs from exploring the parallelism across consecutive tokens, since tokens are generated sequentially rather than simultaneously. Secondly, operating with a single batch size forces LLM to process only one request at a time, thereby eliminating the potential for batch-level parallelism. These constraints significantly restrict the reuse of weight data across multiple tokens, thereby contributing to the extremely low arithmetic intensity observed in single-batch LLM inference.

Figure 3(a) illustrates the arithmetic intensity of different models [32], [36]. Under INT8 quantization, the decode phase of LLM edge inference tasks has an Arithmetic Intensity of merely 2, which is orders of magnitude lower than that observed in the prefill phase and other model inference tasks. This leads to severe memory bandwidth bottlenecks when traditional mobile NPUs are employed. To enhance the efficiency of LLM inference at the edge, Cambricon-LLM leverages on-die processing to minimize data transfer volume, and significantly improve the execution efficiency of the LLM inference during the decode phase, moving from point A to point B.
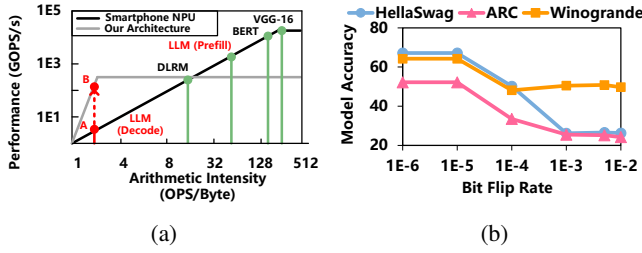
Fig. 3: (a) Roofline model analysis of our architecture and smartphone NPU and (b) the sensitivity test of OPT-6.7B under bit-flip error in flash.

TABLE I: Storage density of DRAM and NAND Flash

Manufacturer	Type	Layers	Storage Density (Gb/mm^2)
SK hynix	Flash	300+	20.00
Samsung	Flash	280	28.50
SK hynix	DDR	1	0.30
SK hynix	LPDDR	1	0.31

B. Why Flash?

Given the large parameter size, LLM inference can greatly benefit from the high storage density offered by flash memory. As indicated in Table I, flash memory boasts a storage density of 10-30 Gb/mm^2, which is two orders of magnitude greater than that of DRAM [29], [30], [34], [57]. This makes flash the best choice for storing large models in space-constrained edge environments. Specifically, a typical 200GB NAND flash chip occupies about 64 mm^2, which is comparable to the area of a smartphone SoC, generally around 100 mm^2. This also proves the feasibility of the hybrid chiplet design proposed by Cambricon-LLM from an area perspective.

Edge inference tasks are minimally impacted by the two primary disadvantages of flash memory in terms of read and write operations. On one hand, flash writes, which involvs programming and erasing, are executed in page-sized and block-sized units respectively. These operations are considerably slower than reads, often by several orders of magnitude. However, for edge-based LLM inference tasks, which solely involve reading weight data from flash without any writing, this slow write speed becomes irrelevant. On the other hand, data stored in flash is extracted from NAND flash memory array to data register in page-sized unit, ranging from 4 to 16KB. This attribute can result in slow fragmented random data access when only a small portion of the page data is needed. Nevertheless, this limitation is inconsequential for LLM edge inference, where data reads are sequential, extensive, and predictable. For example, under INT8 quantization, even the smallest weight matrix of the llama2-7B model is 16MB, leading to the minimal overhead of splitting the model weights into page-sized segments.

C. The Necessity of Error Correction

Flash memory is known for its high error probability and the dominant are retention errors [6], [7]. This error occurs due to electrons leaking from the floating gate of the floating-gate field-effect transistors, leading to alterations in the stored information. The bit error rate of a new 3D TLC NAND chip can reach 1×10^{-4} [80] after hours of retention time. As the flash ages with an increasing number of P/E cycles, the bit error rate can rise to over 1×10^{-2} [7]. To mitigate these errors, SSDs are designed with complex error-correction logic in the controller, such as Low-Density Parity-Check (LDPC)

error-correcting codes. However, due to the substantial area required by the error correction components, they cannot be integrated within the die, resulting in unavoidable error issues for on-die processing flash. Figure 3(b) demonstrates the effect of different error rates on the LLM inference outcome. These errors cannot be neglected, as they diminish the accuracy by over 70%, making the result unreliable. Therefore, to ensure the accuracy of LLM inference, a lightweight error correction module is on demand for on-die processing flash. This module should be compact enough to fit within the die while still ensuring the precision of LLM inference.

IV. Cambricon-LLM Architecture

This section presents the Cambricon-LLM architecture. Specifically, Section IV-A introduces the overview of Cambricon-LLM, Section IV-B presents the design of the Flash die, and Section IV-C describes the detailed design of the Slice Control.

A. Overview

Figure 4 (a) presents the overall Cambricon-LLM architecture, which primarily consists of an NPU and a flash. The flash equips on-die processing capabilities and connects to the NPU through high-speed Die-to-Die (D2D) chiplet links. Additionally, the NPU integrates a flash controller to establish direct communication with the flash, marking a deviation from the designs of traditional NPUs. Both the flash and the NPU contain Processing Elements (PEs) and work collaboratively to compute the GeMV multiplication between the weight matrices and the input vectors. The NPU is equipped with a Special Function Unit (SFU) dedicated to managing specialized functions essential for LLM inference, such as Softmax, sin/cos, and ReLU. There is no SFU embedded in the flash as the on-die processing of these operations provides no tangible benefit and the complexity of implementing such logic on a flash die is prohibitively high.

During single-batch inference of LLMs, the substantial and invariable weights remain in flash memory while the relatively small but frequently updated KV cache (*e.g.* less than 700MB for 70B LLM) is allocated to DRAM. To efficiently process each layer of the LLM, we categorize all LLM operations into three groups and map them onto the hardware components of Cambricon-LLM based on their unique characteristics. As shown in Figure 5, ① All GeMV operations that take the model weight as input are collaboratively executed by NPU and flash (yellow box). We utilize flash for these operations because its on-die processing units are positioned near the weight data and enable exploiting the internal parallelism of the flash. We also incorporate the NPU to prevent underutilization of flash channels, which can occur from the high reduction ratio of these operations when using flash alone. ② Matrix operations
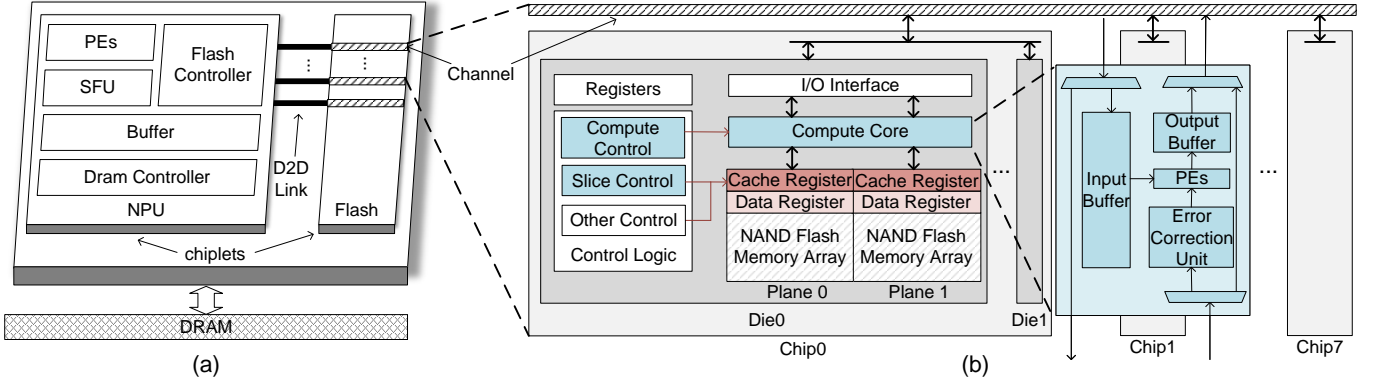
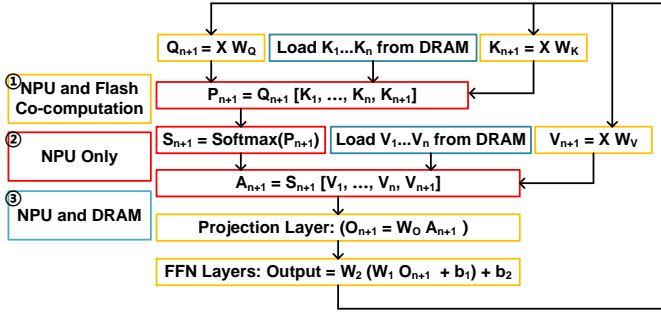Fig. 4: Overall architecture of Cambricon-LLM.

Fig. 5: Compute flow and hardware mapping of an LLM inference layer.

associated with the KV cache are exclusively handled by the NPU (red box). This is because the KV cache is stored in DRAM, which is closer to the PEs of the NPU. ③ Operations involving KV cache loading are managed jointly by the NPU and DRAM (blue box), simply because their functionality necessitates the involvement of both DRAM itself and the DRAM controller on the NPU.

To efficiently handle the GeMV multiplication with the flash die, we introduce a dedicated flash die design and a novel read-compute request, which will be discussed in Section IV-B.

B. Design of Flash Die

To enable the computation capability of the GeMV operations, each flash die in Cambricon-LLM incorporates a novel shared Compute Core and two additional control logics (*i.e.*, the Compute Control and the Slice Control), as depicted in Figure 4 (b). Note that though the shared Compute Core can only be exclusively occupied by a single plane during processing and increases the effort of the computational scheduling, it effectively mitigates the issues of significant area consumption and high error rates. Specifically, the Compute Core is logically complex, encompassing multiple arithmetic units, buffers, and error correction modules. Thus, providing an independent Compute Core for each plane would lead to an unacceptable area consumption. Moreover, the Compute Core generates a considerable amount of heat during calculations, potentially raising the temperature of the flash memory array, which can in turn increase the error rate. The increasing error

rate could impact the inference accuracy when it surpasses the correction capabilities of the on-die Error Correction Unit.

Other than the Compute Core, the additional control logics in Cambricon-LLM help enhance on-die processing capability as well. On the one hand, the Compute Control allows flexible execution of GeMV computations with diverse shapes. On the other hand, the Slice Control plays a crucial role in effectively segmenting the entire read request into smaller slices, which prevents it from continuously occupying the channel and blocking subsequent requests. By efficiently managing the data flow, the Slice Control ensures smooth and uninterrupted processing. More details about the Slice Control will be discussed in Section IV-C.

To better support the on-die computation, we propose a novel read-compute request instruction, supported by the Compute Core consisting of PEs, buffers, and an Error Correction Unit. The read-compute request executes the GeMV multiplication inside each Compute Core as follows: ❶ The input vector is sent to each Compute Core through the flash channel and stored in the input buffer of the Compute Core. ❷ The weight matrix, originally stored in the NAND flash memory array, is fetched into the data register. Since the access of NAND flash memory array data is page-based, the size of the weight matrix for each computation must be the same as the page size. ❸ The weight matrix is then transferred from the data register to the cache register to be used for further computation, leaving the data register empty to serve the next request, and thus form a pipeline format. ❹ The PEs in the Compute Core conduct GeMV multiplication using the matrix from the cache register and the vector from the input buffer. The resulting vector is stored in the output buffer. Notably, the computing power of the Compute Core must match the read speed of the flash memory array. For example, for a certain flash that requires 20us (*tR*) to read a 16KB (page size) weight matrix, the PE must complete 32K operations (INT8) within 20us to avoid throughput delays. This corresponds to a computing power of 1.6 GOPS, which needs approximately two Multiply-Accumulate units (MAC). ❺ When the computation is finished, the result vector is sent back to the NPU through the flash channel.
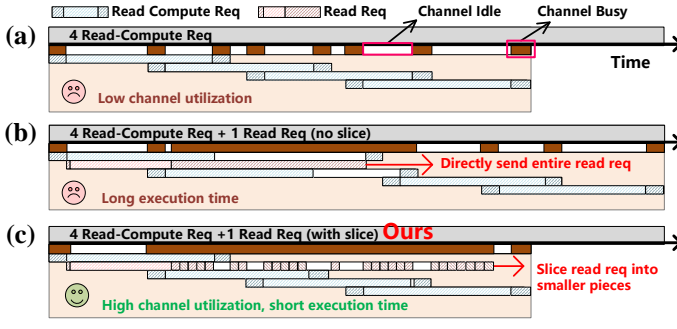
Fig. 6: An example of the pipeline of the requests and the channel state with the three Slice Control strategies, (a) with only read-compute requests, (b) with both read-compute requests and read requests, and (c) with both read-compute requests and sliced read requests (ours).

C. Design of Slice Control

To enhance channel bandwidth utilization, Cambricon-LLM introduces a novel Slice Control within the die, aiming to maximize the utilization of the available bandwidth. The Slice Control achieves this goal in two steps. **First,** the result vector of the read-compute request has a relatively small size compared to the original weight matrix, and their transfer time through the flash channel is negligible compared to the page read time tR, causing a low bandwidth utilization ($\leq 6\%$). To address the low bandwidth utilization with only read-compute requests, the idle plane serves normal read requests to deliver model weights to the NPU when the other plane processes read-compute requests, avoiding the waste of limited channel bandwidth. **Second,** our experiments reveal that transferring read-request data continuously in a page manner through the flash channel is suboptimal, as the prolonged transfer time can not fit in the channel occupancy bubbles between read-compute requests, leading to the blocking of subsequent read-compute requests. To mitigate this problem, we introduce a novel read request slice mechanism that segments a page-sized read request into smaller slices. The read requests are transferred slice by slice during the channel occupancy bubbles until the entire page is transferred so that the channel resources can be better used and the blocking of subsequent requests no longer exists.

Take a simplified configuration as an example, in which one channel connects one chip, with a single die that contains two planes and a shared Compute Core. Figure 6 shows the pipeline of the requests and the channel state with the three Slice Control strategies, *i.e.*, with only read-compute requests (a), with both read-compute requests and read requests (b), and with both read-compute requests and sliced read requests (c). For strategy (a), the continuous issuance of four read-compute requests leads to suboptimal bandwidth utilization, evidenced by the extensive white spaces in the black bars. This inefficiency stems primarily from the mismatch between the flash memory data read times and the result vector transfer times. For strategy (b), one unsliced read request interleave with four read-compute requests, resulting in severe blocking
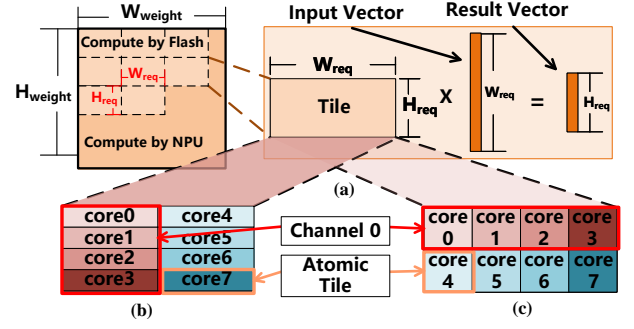

Fig. 7: The hardware-aware tiling strategy in Cambricon-LLM.

of the last two read requests and an extended total execution time. For strategy (c) with the proposed read request slice mechanism, the sliced read requests are interposed among four read-compute requests. This arrangement eliminates the blocking of subsequent read-compute requests and therefore significantly improves the channel utilization, making the execution time align with (a).

V. HARDWARE-AWARE TILING

To optimize the use of available flash channel bandwidth resources, we propose a novel hardware-aware tiling strategy that efficiently allocates the LLM inference workload between the NPU and the flash. This approach sophistically partitions the GeMV workload across both the NPU and the flash, and utilizes the idle periods of the flash channel when processing the read-compute requests to transfer a portion of the weight matrix to the NPU for processing. The hardware-aware tiling strategy works in two steps. **Firstly**, we identify the optimal tile shapes that align best with the hardware specifications of the flash (e.g. page size, channel number, and chip number). **Secondly**, we determine the optimal workload distribution proportions to balance the computation between the NPU and the flash.

A. Identify the Optimal Shape of Tiles

We first identify the optimal shape of the matrix tiles which match the hardware specifications of the flash. As shown in Figure 7 (a), during LLM inference, the whole weight matrix ($H_{weight} \times W_{weight}$) is segmented into smaller tiles ($H_{req} \times W_{req}$). Each tile is processed collaboratively by all Compute Cores in the flash, corresponding to a single read-compute request. The efficiency of LLM inference heavily relies on the shape of shapes, as it determines the lengths of input and result vectors, consequently impacting the overall volume of data transmitted through the flash channels. Our goal is to identify the shape of the tile to minimize the overall data transfer.

Consider a simple flash setup with 2 channels, each connected to 4 Compute Cores (cores 0-3 on channel 0, and cores 4-7 on channel 1). To fully utilize all computational resources in the flash, each tile will be evenly distributed to all 8 Compute Cores, with each Compute Core handling a segment named atomic tile, as described in Figure 7 (b). Each colored block represents an atomic tile computed by a specific

Compute Core, and blocks of the monochromatic color indicate allocation to the same channel. Define $channel_{num}$ as the total number of channels in the flash, and $ccore_{num}$ as the total number of Compute Cores connected to each channel. The tile is split vertically by channels into 2 segments, with each channel handling a segment of shape ($H_{req} \times \frac{W_{req}}{channel_{num}}$). Given that $ccore_{num}$ Compute Cores are connected to each channel, each Compute Core handles an atomic tile of shape ($\frac{H_{req}}{ccore_{num}} \times \frac{W_{req}}{channel_{num}}$).

Each atomic tile is independently computed by one Compute Core and the total data transfer volume through the flash channel equals the size of the input vector plus the size of the result vector. Given $channel_{num} \times ccore_{num}$ Compute Cores on the flash, we can get a naive data transfer volume of $Trans = ccore_{num} \times W_{req} + channel_{num} \times H_{req}$. However, we observe that the four cores on the same channel (*e.g.*core 0/1/2/3) share mutual input vectors. Therefore, **broadcast operations** can be employed to simultaneously send input vectors to all Compute Cores on the same channel. Those input vectors are stored in the input buffer of each Compute Core for further computation. This avoids repeatedly sending the same input vector multiple times through the flash channel and therefore reduces the total transferred data volume to:

$$Trans = W_{req} + channel_{num} \times H_{req}.$$

After deriving the expression for the total data volume transferred, our next objective is to determine the optimal tile dimensions (H_{req} and W_{req}) that minimize $Trans$. Given that the size of each atomic tile matches the page size, the following relationship is established under INT8 quantization:

$$H_{req} \times W_{req} = channel_{num} \times ccore_{num} \times pagesize.$$

Consequently, the formula for $Trans$ represents a standard AM-GM inequality, allowing us to readily compute the minimal data transfer size:

$$\min\{Trans\} = 2 \times \sqrt{W_{req} \times H_{req} \times channel_{num}}$$
$$= 2 \times channel_{num} \times \sqrt{ccore_{num} \times pagesize}.$$

This minimum is attained when:

$$H_{req}* = \sqrt{ccore_{num} * pagesize},$$
$$W_{req}* = channel_{num} * \sqrt{ccore_{num} * pagesize}.$$

For now, we identify the optimal tile shape for the flash, which depends solely on the number of channels, the number of Compute Cores, and the page size. During LLM inference, we tailor each weight matrix into this specific shape to maximize the inference speed.

It is important to note that this analysis is based on the splitting scheme shown in Figure 7 (b) There is, however, an alternative splitting scheme illustrated in Figure 7 (c). In this alternative approach, the input vector of atomic tiles shows no chance of data reuse, resulting in a total data transfer volume of: $Trans_2 = core_{num} \times W_{req} + channel_{num} \times H_{req}$ This expression also follows the AM-GM inequality, but

the minimum, $2 \times \sqrt{W_{req} \times H_{req} \times channel_{num} \times ccore_{num}}$, is larger than $\min\{Trans\}$, rendering this scheme less optimal.
.

B. Determine Optimal Workload Distribution Proportions

After the optimal tiling shape is determined, we then need to determine the proportion of workload allocated to NPU and flash, respectively. The primary goal is to balance the execution times of the flash and the NPU. While the execution time on the flash depends on the read-compute requests, the execution time on the NPU is dominated by data transfer time (read requests), given its considerable computing power and the modest computational demand of LLM single-batch inference. Therefore, optimizing the workload distribution requires a careful balance between the execution times of read-compute requests and read requests. To achieve this, we first estimate the execution time for a single read-compute request and a single read request, then determine the proportion that balances the total execution time for all read-compute requests and read requests.

For read-compute requests, neglecting pipeline startup latency, the execution time t_{rc} is the sum of the input data transmission times through the flash channel and the read operation latency from the flash memory array:

$$t_{rc} = t_R + \frac{W_{req}}{channel_{num} \times bw_{channel}}.$$

Where t_R is the read operation time, W_{req} is the tile width, $channel_{num}$ is the number of channels, and $bw_{channel}$ is the channel bandwidth. To determine the execution time of read request, we first calculate the channel utilization rate for read-compute requests, which informs us how much bandwidth remains for the read requests.

$$rate_{rc} = \frac{H_{req} + \frac{W_{req}}{channel_{num}}}{t_R \times bw_{channel}}.$$

Then, the execution time for a read request (t_r) can be estimated as:

$$t_r = \frac{pagesize}{(1 - rate_{rc}) \times bw_{channel}}.$$

The optimal workload distribution is achieved when the execution times for read and read-compute requests are equal. Denote the proportion for read compute workload as α, then:

$$\alpha = \frac{t_r}{t_r + t_{rc}}.$$

Based on the distribution proportion α, and the optimal tile shape H_{req} and W_{req}, we devise a tiling strategy for all GeMV multiplication operations during LLM inference. As depicted in Figure 7 (a), an α proportion of the weight matrix ($H_{weight} \times W_{weight}$) is assigned to flash in a tiled manner, each tile sized ($H_{req} \times W_{req}$). In this figure, there are 8 tiles being allocated to flash in total and the remainder weight matrix is transferred through flash channels and processed on the NPU.
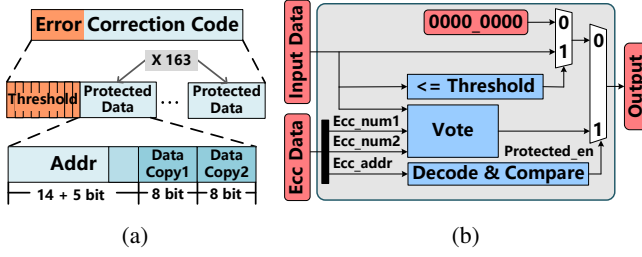
Fig. 8: (a) The structure of our proposed ECC and (b) the design of on-die error correction unit in Cambricon-LLM.

VI. Error Correction Mechanism

To address the high error rates of the flash memory and ensure accurate LLM inference, we propose an efficient on-die error correction algorithm and a lightweight hardware implementation. **Our key observation** is that when dealing with LLM models containing over 7 billion (7B) parameters, the model's accuracy is particularly affected by a small subset of outliers, which account for less than 0.1% of the total parameter values [12], [22]. These outliers have significantly larger absolute magnitudes compared to the regular parameter elements. Based on this insight, we develop an outlier-oriented error correction code (ECC) and a hardware-friendly algorithm to enable on-die error correction. The ECC is stored in the spare area of flash memory that is originally designated for ECC storage and occupies 1664 bytes for a 16KB page.

The error correction operates on two fronts. Firstly, it safeguards outliers in the original weight matrix against flash errors. Secondly, it prevents normal values from being transformed into outliers due to bit-flip errors in flash memory. For the outliers protection, we identify the top 1% largest values within the entire page, and then store their positions and N copies of their values into the error correction code (ECC) for protection (N is an even number). During the inference of LLM, the ECC is decoded on die and a bit-wise majority vote is conducted between the N copies and the original data.

Take the example of an outlier valued 64(0b'0100_0000) and $N = 2$ (meaning the ECC contains 2 copies of the outlier value). If the three recorded instances are 0110_0000, 0100_0000, and 0100_0000 respectively, then the output value after decoding will be 0100_0000, thus preserving the original outlier data under flash memory error. TO quantify the protection capability of this mechanism, we set the single-bit flip rate of the flash array to be x. Only if more than $N/2+1$ of the $N+1$ instances of the outlier copies are flipped, will the output be flipped. Therefore, the flip rate of protected outliers can be diminished to:

$$
\begin{aligned}
f_{prot} &= \sum_{i=N/2+1}^{N+1} \binom{N+1}{i} \times x^i (1-x)^{N+1-i} \\
&\approx \binom{N+1}{N/2+1} \times x^{N/2+1} \\
&= \frac{(N+1)!}{(N/2+1)! \times (N/2-1)!} \times x^{N/2+1}.
\end{aligned}
$$

If $N = 2$ and the original flip rate is 1×10^{-4}, then the protected flip rate $f_{prot} = 3x^2 = 3 \times 10^{-8}$.

To prevent normal values from being affected, while encoding the top 1% largest values of the page, we also record the smallest among these large values as a threshold. During decoding, if any value exceeds the threshold yet is not marked in the ECC as an outlier, it is inffered to be generated by flash memory errors. To prevent such fake large values from compromising the accuracy, we trunk these values to zero.

Additionally, we propose the detailed ECC structure and a lightweight on-die error correction hardware implementation to process the algorithm. As shown in Figure 8(a), the ECC data structure stores the threshold at the beginning for multiple copies (e.g., 9 copies) to ensure its safety since it is crucial. Following the threshold, the 1% protected values in the entire page are stored. For a 16KB page with the weight format as INT8, there are a total of 16384 data points in a page, thus a total of 163 large values are protected. For each protected large value, its address is stored along with two copies of the data. When each page contains 16,384 elements, a total of 14 bits are required to describe the address of each element. As the address itself can also be prone to errors, each address is accompanied by a 5-bit private error-correcting code for their safety, utilizing the format of Hamming code. This encoding format has a low hardware decoding overhead and can be efficiently completed on-die. If a 1-bit error occurs in the address, it will be corrected by the on-die decoder. If a 2-bit error occurs, the protected value will be discarded and treated as unprotected. The total size of the ECC for each page is $8 \times 9 + (14 + 5 + 8 \times 2) \times 163 = 722B$, which is less than the size of the spare space for each page (1664B).
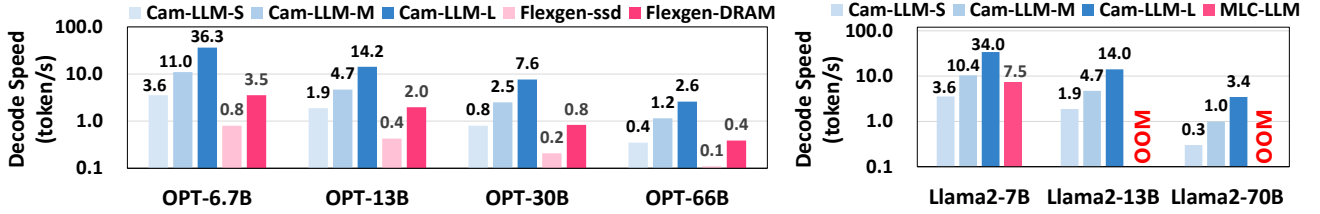
Figure 8(b) depicts the hardware implementation of the on-die Error Correction Unit. For each data retrieved from flash memory, it first evaluates whether the current input data falls within the protected range by comparing the address of current input data to the address of protected outliers stored in ECC. If the input data is identified as protected, error correction is enabled through a voting mechanism utilizing the two copies stored in ECC along with the input data itself. Otherwise, if the input data is identified as unprotected, its value should be no larger than the threshold. Following this principle, the on-die Error Correction Unit compares the unprotected data with the threshold and directly output it if the unprotected data is smaller than the threshold. Otherwise, this unprotected data is deemed an anomalous value likely resulting from flash memory error and is trunked to zero. In summary, with the efficient on-die error correction algorithm and its lightweight hardware implementation, Cambricon-LLM effectively neutralizes the high error rates of flash memory, preserving the accuracy of LLM inference.

VII. Experiment Setup

A. Hardware Configuration

We compare the performance of Cambricon-LLM with two mainstream LLM inference frameworks named Flexgen [60] and MLC-LLM [63]. Flexgen excels in offloading workloads

(a) Performance of Cambricon-LLM and Flexgen   (b) Performance of Cambricon-LLM and MLC-LLM

Fig. 9: End-to-end performance of Cambricon-LLM compared to (a) Flexgen and (b) MLC-LLM.

TABLE II: Configurations of Cambricon-LLM

	Cambricon-LLM-S	Cambricon-LLM-M	Cambricon-LLM-L
Quantization	8bit	8bit	8bit
Channel	8	16	32
Chip	2	4	8
Flash Memory	2 die per chip, 2 plane and 1 compute core per die, 1000MT/s, 8 bit channel bus, page size = 16KB		
Time Parameters	tR = 30us		

TABLE III: Configurations of baselines

	Flexgen-SSD	Flexgen-DRAM	MLC-LLM
Quantization	8bit	8bit	4bit
Weight Offloading	SSD	DRAM	DRAM
Hardware Configuration	AMD EPYC 7742 CPU, A100 80G GPGPU, Intel NVMe SSD, 128GB DRAM		Qualcomm Snapdragon 8 Gen 2

TABLE IV: Area and power overhead of compute core

	Area (um^2)	Power (uw)
Error Correction Unit	496.4	0.4
PEs	562.0	343.6
Input Buffer and Output Buffer	58755.1	1591.7
Total Compute Core	39813.5	1935.6
Overhead	1.2%	4.5%

MLC-LLM: We test the performance of MLC-LLM on smartphones equipped with the Qualcomm Snapdragon 8 Gen 2 Soc. However, it is important to point out that the Llama2-7B model supported on MLC-LLM is based on 4-bit round-to-nearest (RTN) quantization.

B. Benchmarks

We evaluate the performance of Cambricon-LLM under on widely used LLMs such as OPT [78] and Llama2 [65], which have parameter sizes ranging from 6.7B to 70B. We conduct model accuracy experiments on popular datasets including HellaSwag [77], Arc [72], and WinoGrande [55]. For the accuracy tests, we utilize the most advanced offline quantization framework, Smoothquant [70], to quantize model weights into INT8. Additionally, we construct flash error models of varying intensities using PyTorch and inject them into quantized model weight.

C. Area and Power Estimation

We developed the computational core of Cambricon-LLM using Verilog HDL and synthesized it through the Design Compiler utilizing the TSMC 65nm process technology node. As indicated in Table IV, the primary contributors to overhead are input buffer and output buffer. These buffers serve the purpose of storing input and output vectors, with a combined capacity of 2KB, which suffices for the majority of applications. The area overhead for the compute core is 1.2% and the power overhead is 4.5%, both of which are within acceptable limits.

VIII. EXPERIMENT RESULT

A. End-to-end performance

This section presents the end-to-end decode speed of Cambricon-LLM compared to the baselines. Figure 9(a) shows the decode speed of Cambricon-LLM and Flexgen for OPT models with varying parameter sizes. **Cambricon-LLM-L** achieves rapid speed across all sizes of OPT models. On the OPT-66B model, its decode speed reaches 2.59 tokens/s,
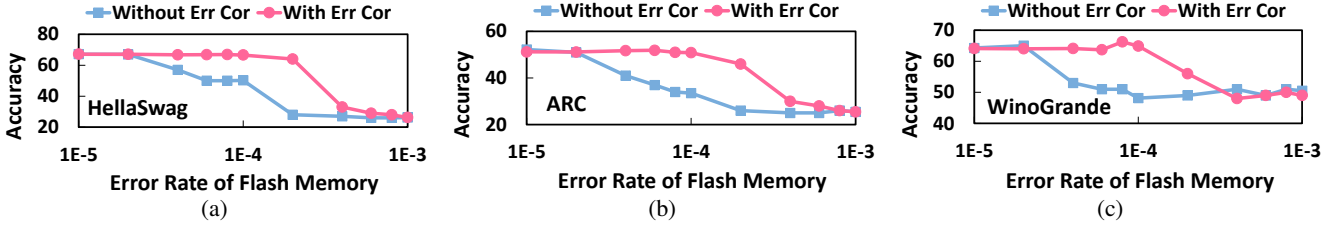
to SSDs, while MLC-LLM is optimized for LLM inference on mobile devices. We present the hardware configurations of Cambricon-LLM and the baselines as follows.

Cambricon-LLM: We use the well-known SSDsim simulator [26] for Flash simulation and build a cycle-accurate simulator in C for NPU simulation. For the Flash simulation, we expand the existing Read/Write command support in SSDsim to include Read Compute commands for in-storage computing. As shown in Table II, we implement three versions of Cambricon-LLM and tailor each setting to different flash resources, named Cambricon-LLM-S, Cambricon-LLM-M, and Cambricon-LLM-L. For the NPU, we utilize a 16x16 systolic array, capable of delivering 2 TOPS of computational power at a 1 GHz clock speed. The NPU is interfaced with LPDDR5X DRAM, which provides approximately 40GB/s of bandwidth. The DRAM is exclusively used for storing the KV cache, and a capacity of 700MB suffices for the needs of a 70B LLM under single batch inference. All test results are based on INT8 quantization, offline quantization techniques ensure model accuracy under this quantization level. It should be noted that all quantization techniques are orthogonal to our architectural optimizations, Cambricon-LLM will proportionally benefit from the development of more aggressive quantization strategies, such as 4/2 bit.

Flexgen: As shown in Table III, We deploy Flexgen on a server and test its performance under different configurations. When using the Flexgen framework, we place both attention and activation on the GPU's HBM and place the model weights on the system DRAM (Flexgen-DRAM) and NVMe SSD (Flexgen-SSD) respectively. Since Flexgen only supports OPT, we only test its performance on the OPT series model.

Fig. 10: Accuracy evaluation of the proposed error correction mechanism.

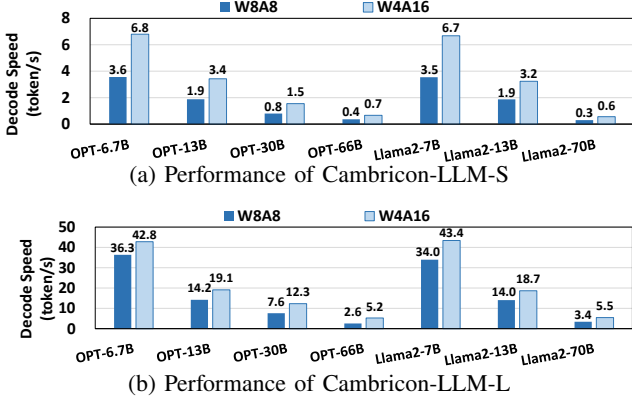(a) Performance of Cambricon-LLM-S

(b) Performance of Cambricon-LLM-L

Fig. 11: Performance of Cambricon-LLM under W4A16 quantization

which is 22.1× faster than Flexgen-SSD and 6.8× faster than Flexgen-DRAM. On the OPT-6.7B model, the inference speed of Cambricon-LLM-L even reaches 36.34 tokens/s, which is 44.8× faster than Flexgen-SSD and 10.3× faster than Flexgen-DRAM. **Cambricon-LLM-M** achieved a speed comparable to Flexgen-DRAM across various tasks, with 10.96/4.68/2.50/1.15 tokens/s for the OPT-6.7B/13B/30B/66B models, respectively. **Cambricon-LLM-S**, the configuration with minimal flash resources, attained a speed of 3.56 tokens per second on the OPT-6.7B model, achieving a smooth operational standard and outperforming Flexgen-SSD by 8.9×.

Figure 9(b) illustrates the decoding speed of Cambricon-LLM and MLC-LLM using Llama2 models with varying parameter sizes. MLC-LLM places all LLM weights entirely in DRAM for inference, thus it is restricted to supporting only up to 7B parameters. On Llama2-13B and 70B, it encounters out-of-memory issues. When utilizing the Qualcomm 8 Gen 2 processor, MLC-LLM achieves a decoding speed of 7.58 tokens/s on Llama2-7B, which is higher than Cambricon-LLM-S's speed of 3.55 tokens/s. This performance disparity is due to MLC-LLM's use of 4-bit quantization, which results in model size reduction but significant precision loss compared to the 8-bit quantization used by Cambricon-LLM-S. Theoretically, employing 4-bit quantization in Cambricon-LLM-S as well could improve the inference speed to match the MLC-LLM.

B. Performance under 4 bit Quantization

To demostrate how our Cambricon-LLM architecture can benifit from lower bit quantization techniques, we conducted experiment evaluating the performance of Cambricon-

LLM with 4 bit quantization for weight and 16 bit quantization for activation (W4A16). In comparison to the defaut 8 bit quantization (W8A8), this W4A16 quantization of model weight requires less bandwidth. Figure 11 presents a comparison of W8A8 and W4A16 quantizations across both Cambricon-LLM-S and Cambricon-LLM-L configurations. On average, W4A16 quantization yields performance improvements of 85.3% for Cambricon-LLM-S and 47.9% for Cambricon-LLM-L. Additionally, we observed that larger performance improvements occur in larger LLMs, which face more severe memory constraints when loading model weights. Reducing the bit-width of model weights significantly boosts their performance.

C. Ablation Study

This section analyzes the performance improvement with the two main techniques of Cambricon-LLM, *i.e.*, read request slicing and hardware-aware tiling. This section also test the affect of different tile sizes

For read request slicing, we use Cambricon-LLM-S configuration and compare the full-featured Cambricon-LLM with a simplified version without this feature. In the simplified version, read requests are sent in a continuous, unsegmented manner. Figure 12 shows the results for decoding speed and channel utilization on different LLM configurations. The results show that the read request slicing achieves a speed up ranging from 1.6x to 1.8x and increases bandwidth utilization by 31.6% to 41.4%. This improvement is due to the sliced read requests not blocking the execution of read compute requests. Read request data will only occupy the channel slice by slice during the bubbles in read compute requests.

For the hardware-aware tiling, we again use the Cambricon-LLM-S configuration and compare the full-featured Cambricon-LLM-S with a version that did not employ the tiling. Without tiling, all operations involving the multiplication of weight matrices and input vectors are completed by Flash, with no offloading to NPU. Figure 14 displays the differences in inference speed and channel utilization on different LLMs. The results show that hardware-aware tiling can accelerate LLM inference by 1.3x to 1.4x and increase channel utilization by 76.2% to 88.9%. This is because the tiling strategy utilizes idle channel bandwidth to transfer some weights to the NPU for computation, thereby reducing the workload of on-die processing on the flash.

We conducted further experiments to assess how tile sizes influence the performance of Cambricon-LLM. In these ex-
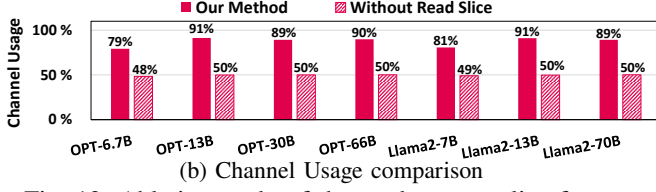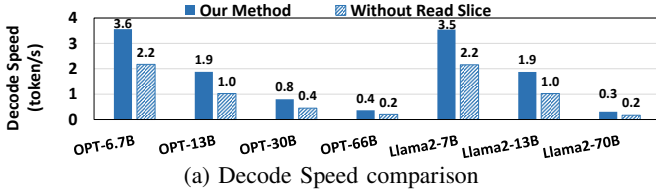
(a) Decode Speed comparison

(b) Channel Usage comparison

Fig. 12: Ablation study of the read request slice feature.

(a) Decode Speed Comparison

(b) Channel Usage Comaprison

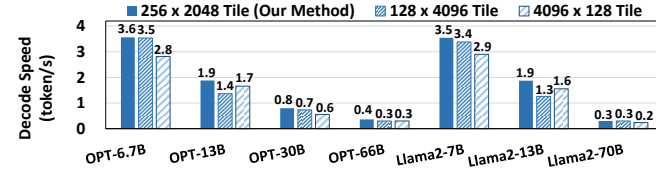Fig. 14: Ablation study of the hardware-aware tiling feature.

Fig. 13: Performance of Cambricon-LLM under varied tile sizes.

periments, We facilitated collaborative operation between the NPU and flash to handle GEMV operations, and explored the effects of multiple sub-optimal tile sizes on performance. We selected the Cambricon-LLM-S configuration with an theoretically optimal tile size of 256×2048 (Height × Width). Additionally, we evaluated the performance of two other configurations with tile sizes of 128×4096 and 4096×128. As illustrated in Figure 13, the optimal tile size of 256×2048 outperforms the 128×4096 configuration by 17.5% and the 4096×128 configuration by 24.7%.

D. On-Die Error Correction Mechanism

We evaluate the sensitivity of the OPT-6.7B model to flash memory errors and the effectiveness of the proposed on-die error correction mechanism on three datasets: HellaSwag, ARC, and WinoGrande. As shown in Figure 10, without the Error Correction Unit, the accuracy of the LLM begins to significantly decrease when the error rate reaches 1×10^{-5}. When the error rate further achieves 2×10^{-4}, the accuracy falls to about 40% of its original level, making the output unreliable. On the other hand, with Cambricon-LLM's on-die Error Correction Unit enabled, the LLM can maintain 92%~95% of its original accuracy at an error rate of 2×10^{-4}, providing 2.3× protection capabilities compared to baseline.

However, the protection capability of Cambricon-LLM also has its limits. When the error rate exceeds 8×10^{-4}, the accuracy of LLM still drops to an unusable level even with the Error Correction Unit enabled. This is because the proposed error correction mechanism only protects outliers from being changed and prevents those trivial values below the threshold from flipping into fake outliers. While it offers no protection for intermediate and small values that do not reach the
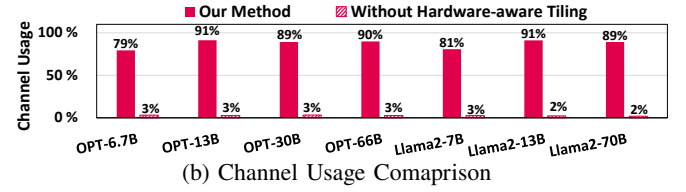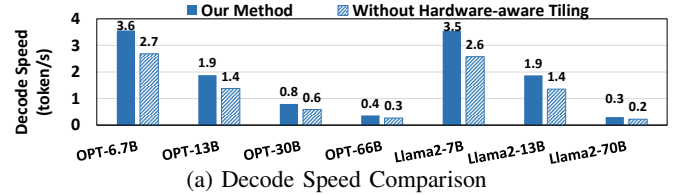
threshold before and after potential bit flips. It is the extensive flipping of these intermediate and small values that leads to the significant degradation in LLM accuracy.

E. Scalability Study

To assess the scalability of Cambricon-LLM, we evaluate the impact of flash channel numbers and chip numbers on the decode speed of Cambricon-LLM on OPT-6.7B/13B/30B.

When examining the effect of the number of chips per channel, we set the number of channels to 8 and increased the number of chips mounted on each channel from 1 to 128. As shown in 15, the decode speed increases rapidly with the number of chips at the beginning, but the growth gradually slows down. This slowdown is attributed to the excessive number of chips, which prevents the model weight from being effectively distributed across all chips. Even with an increased number of chips, many remained idle, yielding no performance gains. As depicted in 15, the utilization of the channels noticeably decreases when there are too many chips on a channel. This decrease is due to an increase in the number of compute cores, leading to more on-die computational power in the flash. Consequently, more computations are allocated to the on-die resources, reducing the computations assigned to the NPU, and thus channels are no longer required to transmit the weight matrix, lowering their utilization.

During the evaluation of how channel count influences performance, we set the number of chips per channel to 4 and incrementally increase the number of channels from 1 to 64. As shown in Figure 15, Cambricon-LLM's performance steadily increases with the number of channels while the channel utilization slowly declines within the test range. This demonstrates good scalability of Cambricon-LLM performance with increasing channel numbers. As the maximum number of flash channels increases, the performance of Cambricon-LLM is expected to improve further. However, it must be noted that since Cambricon-LLM employs chiplet technology to link the NPU and Flash, the number of channels is also constrained by the chiplet fabrication technology and the geometric dimensions. Moreover, in order to accommodate a greater number of channels, the buffer on the NPU needs
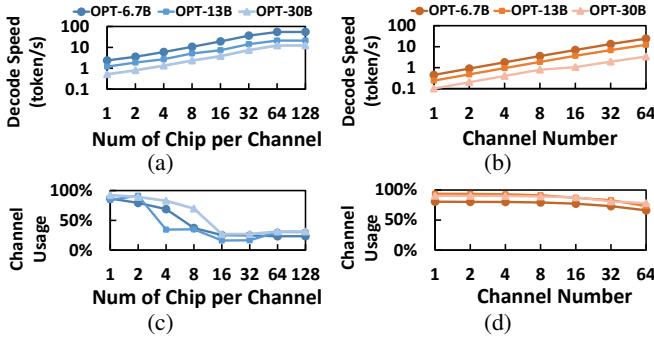
Fig. 15: Scalebility evaluation of channel and chip number.

to be proportionally expanded to hold a larger amount of data from the flash. However, this enlargement of the buffer will result in increased area consumption.

F. Data Transfer and Power Consumption

We evaluate the data transfer size and energy consumption of one token during inference for Cambricon-LLM-S and Flexgen-SSD across various models. The results are shown in Figure 16. Cambricon-LLM-S transfers $9.7\times$ to $11.6\times$ less data Flexgen-SSD. This substantial reduction can be attributed to two primary factors. Firstly, the NPU within Cambricon-LLM has direct access to the data stored in Flash, which eliminates the need for transferring data from Flash to DRAM. Secondly, the Flash in Cambricon-LLM possesses in-storage computing capabilities, allowing for data to be processed and its size reduced within the flash die before transmission. Furthermore, the total energy consumed by Cambricon-LLM-S for data transfer per token is only 67% of that used by Flexgen-SSD. This reduction in energy consumption is primarily due to the decreased size of data transfer and the incorporation of chiplet technology, which reduces the overhead associated with data transmission between NPU and Flash.

G. Cost Analysis

This section compares the cost of Cambricon-LLM with traditional architecture. To support the inference of 70B LLM under 8 bit quantization, a minimum storage of 80 GB is needed. Traditional architecture uses DRAM to store all model weights and KV cache while Cambricon-LLM leverages considerably cheaper flash memory to store model weights, reserving DRAM solely for KV cache. As shown in Table V, Cambricon-LLM offers a cost advantage, being $150.01 cheaper than the traditional architecture.

However, it is essential to acknowledge two additional costs associated with Cambricon-LLM that are not captured in Table V. Firstly, the integration of new logic into flash die incurs extra costs. Nonetheless, this additional logic is relatively simple and area overhead is minimal, redering the associated cost negligible. Secondly, the adoption of chiplet technology in Cambricon-LLM also contributes to overall costs. While specific open-source data on chiplet technology costs are unavailable, recent research on the cost model of chiplets [19] suggests that expenses related to the die-to-die interface and packaging should be less than 15% of the
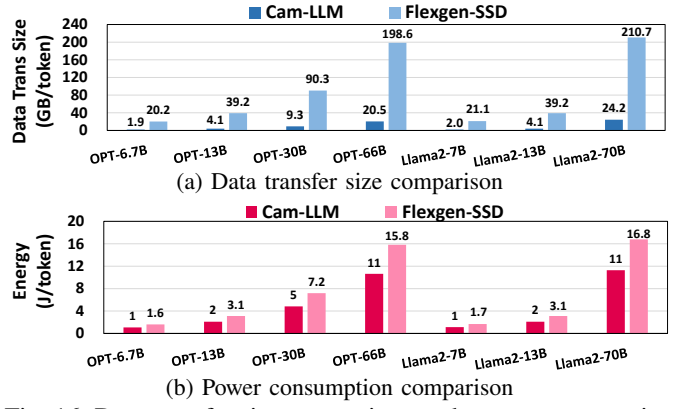

(a) Data transfer size comparison

(b) Power consumption comparison

Fig. 16: Data transfer size comparison and power consumption comparison of Cambricon-LLM -S and Flexgen-SSD.

raw chip costs, estimated not to exceed $100 in the case of Cambricon-LLM.

IX. RELATED WORK

LLM acceleration. There exists a number of prior works to accelerate LLMs. Some works [8], [37], [47], [59], [62], [74] focus on huge batch size LLM inference with substantial hardware resources, while others aim to maximize the use of limited available GPU resources [10], [11]. Additional studies take advantage of the sparsity of attention matrices to make software optimization [3], [67], [76], design new architectures [17], [18], [24], [42], [43], [58] and implement FPGA deployment [38], [52], [53], [73], [79]. However, none of these approaches have the potential to enable the inference of a 70B LLM on smartphones due to DRAM size limitations. To address DRAM constraints, some works have proposed offloading strategies to flash-based SSDs [54], [60], though limited bandwidth slows down the inference speed. The "LLM in a Flash" [1] method attempts to overcome this by exploiting the sparsity of LLMs to reduce data transfer volumes; however, it still requires substantial DRAM and fails to utilize the inherent parallelism of flash memory effectively. Furthermore, numerous quantization-based approaches [13], [20], [22], [40], [41], [70], [75] can also help with the deployment of LLM on edge devices since they can reduce the total weight size. Notably, these quantization methods are all orthogonal to Cambricon-LLM.

In-Storage processing. In-storage computing (ISC) has been widely used for many applications such as data search and analytics [15], [35], [44], [64], [81], DNN [23], [33], [39], [69], biological information processing [45], [71] and graph processing [28], [46]. However, all of these works add the computing logic near the controller, failing to explore the potential of processing on flash dies.

On-die Processing. Only a few works have proposed the design of adding logic on flash dies. Deepstore [44] equipped flash die with a systolic array for query searching, while the logic is too complex to be feasible. Optimstore [31] and [68] adopt on-die logic to accelerate DNN training and large-scale GNN, respectively. However, both of these designs ignore the

TABLE V: Cost of Cambricon-LLM and traditional architecture to support inference of 70B LLM under 8 bit quantization.

	Cambricon-LLM		Traditional Architecture	
	Count	Cost ($)	Count	Cost ($)
DRAM (GB)	2	4.87	80	194.68
Flash (GB)	80	38.80	0	0.00
Total Price	N/A	43.67	N/A	194.68

importance of on-die error correction, making them unusable for LLM inference tasks. Besides, the high reduction ratio of LLM single-batch inference leads to server low flash channel utilization, making them not suitable for this task.

X. Conclusion

We propose Cambricon-LLM, a chiplet-based architecture consists of an NPU and a flash to enable the on-device inference of 70B LLMs. The chiplet design and the direct access between the NPU and the flash ensure minimal energy consumption during data movement. The flash is provided with on-die processing capabilities and works collaboratively with the NPU to handle GeMV multiplication through the proposed hardware-aware tiling strategy. To deal with the high error rate of flash memory, we design a lightweight on-die Error Correction Unit in the flash to guarantee reliability. Cambricon-LLM can conduct inference of 70B LLM at a speed of 3.4 token/s, which is over $22\times$ faster than the state-of-the-art flash-offloading framework.

XI. Acknowledgment

References

[1] K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. D. Mundo, M. Rastegari, and M. Farajtabar, "Llm in a flash: Efficient large language model inference with limited memory," 2024.

[2] E. Almazrouei, H. Alobeidli, A. Alshamsi, A. Cappelli, R. Cojocaru, M. Debbah, Étienne Goffinet, D. Hesslow, J. Launay, Q. Malartic, D. Mazzotta, B. Noune, B. Pannier, and G. Penedo, "The falcon series of open language models," 2023.

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

[4] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[5] M. Brysbaert, "How many words do we read per minute? a review and meta-analysis of reading rate," *Journal of memory and language*, vol. 109, p. 104047, 2019.

[6] Y. Cai, S. Ghose, E. F. Haratsch, Y. Luo, and O. Mutlu, "Error characterization, mitigation, and recovery in flash-memory-based solid-state drives," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1666–1704, 2017.

[7] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Crista, O. S. Unsal, and K. Mai, "Error analysis and retention-aware error management for nand flash memory," *Intel Technology Journal*, vol. 17, no. 1, 2013.

[8] L. Chen, Z. Ye, Y. Wu, D. Zhuo, L. Ceze, and A. Krishnamurthy, "Punica: Multi-tenant lora serving," *arXiv preprint arXiv:2310.18547*, 2023.

[9] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel, "Palm: Scaling language modeling with pathways," 2022.

[10] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[11] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," in *Advances in Neural Information Processing Systems*, 2022.

[12] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "Llm.int8(): 8-bit matrix multiplication for transformers at scale," *ArXiv*, 2022.

[13] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[14] 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.

[15] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt, "Query processing on smart ssds: Opportunities and challenges," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013.

[16] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence, "Palm-e: An embodied multimodal language model," 2023.

[17] H. Fan, S. I. Venieris, A. Kouris, and N. Lane, "Sparse-dysta: Sparsity-aware dynamic and static scheduling for sparse multi-dnn workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023.

[18] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 30, no. 11, pp. 1573–1586, 2022.

[19] Y. Feng and K. Ma, "Chiplet actuary: a quantitative cost model and multi-chiplet architecture exploration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[20] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[21] S. K. Gonugondla, M. Kang, Y. Kim, M. Helm, S. Eilert, and N. Shanbhag, "Energy-efficient deep in-memory architecture for nand flash memories," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018.

[22] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023.

[23] S. Gupta, J. Morris, M. Imani, R. Ramkumar, J. Yu, A. Tiwari, B. Aksanli, and T. Š. Rosing, "Thrifty: Training with hyperdimensional computing across flash hierarchy," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020.

[24] 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.*, "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[25] P. Hämäläinen, M. Tavast, and A. Kunnari, "Evaluating large language models in generating synthetic hci research data: a case study," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, 2023.

[26] Y. Hu, H. Jiang, D. Feng, L. Tian, S. Zhang, J. Liu, W. Tong, Y. Qin, and L. Wang, "Achieving page-mapping ftl performance at block-mapping ftl cost by hiding address translation," in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[27] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei, "Sˆ3: Increasing gpu utilization during generative inference for higher throughput," in *Advances in Neural Information Processing Systems*, 2023.

[28] S.-W. Jun, A. Wright, S. Zhang, S. Xu *et al.*, "Grafboost: Using accelerated flash storage for external graph analytics," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[29] W. Jung, H. Kim, D.-B. Kim, T.-H. Kim, N. Lee, D. Shin, M. Kim, Y. Rho, H.-J. Lee, Y. Hyun *et al.*, "13.3 a 280-layer 1tb 4b/cell 3d-nand flash memory with a 28.5 gb/mm2 areal density and a 3.2 gb/s high-speed io rate," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, 2024.

[30] B. Kim, S. Lee, B. Hah, K. Park, Y. Park, K. Jo, Y. Noh, H. Seol, H. Lee, J. Shin *et al.*, "28.2 a high-performance 1tb 3b/cell 3d-nand flash with a 194mb/s write throughput on over 300 layers," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023.

[31] J. Kim, M. Kang, Y. Han, Y.-G. Kim, and L.-S. Kim, "Optimstore: In-storage optimization of large scale dnns with on-die processing," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[32] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney *et al.*, "Full stack optimization of transformer inference: a survey," *arXiv preprint arXiv:2302.14017*, 2023.

[33] S. Kim, Y. Jin, G. Sohn, J. Bae, T. J. Ham, and J. W. Lee, "Behemoth: a flash-centric training accelerator for extreme-scale {DNNs}," in *19th USENIX Conference on File and Storage Technologies (FAST 21)*, 2021.

[34] W. Kim, C. Jung, S. Yoo, D. Hong, J. Hwang, J. Yoon, O. Jung, J. Choi, S. Hyun, M. Kang *et al.*, "A 1.1 v 16gb ddr5 dram with probabilistic-aggressor tracking, refresh-management functionality, per-row hammer tracking, a multi-step precharge, and core-bias modulation for security and reliability enhancement," in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023.

[35] G. Koo, K. K. Matam, T. I, H. K. G. Narra, J. Li, H.-W. Tseng, S. Swanson, and M. Annavaram, "Summarizer: trading communication with computing near storage," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, 2017.

[36] J. Kosaian and K. Rashmi, "Arithmetic-intensity-guided fault tolerance for neural network inference on gpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021, pp. 1–15.

[37] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

[38] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020.

[39] S. Li, F. Tu, L. Liu, J. Lin, Z. Wang, Y. Kang, Y. Ding, and Y. Xie, "Ecssd: Hardware/data layout co-designed in-storage-computing architecture for extreme classification," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023.

[40] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for llm compression and acceleration," 2024.

[41] S.-y. Liu, Z. Liu, X. Huang, P. Dong, and K.-T. Cheng, "Llm-fp4: 4-bit floating-point quantized transformers," *arXiv preprint arXiv:2310.16836*, 2023.

[42] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021.

[43] S. Lu, M. Wang, S. Liang, J. Lin, and Z. Wang, "Hardware accelerator for multi-head attention and position-wise feed-forward in the transformer," in *2020 IEEE 33rd International System-on-Chip Conference (SOCC)*, 2020.

[44] V. S. Mailthody, Z. Qureshi, W. Liang, Z. Feng, S. G. De Gonzalo, Y. Li, H. Franke, J. Xiong, J. Huang, and W.-m. Hwu, "Deepstore: In-storage acceleration for intelligent queries," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[45] N. Mansouri Ghiasi, J. Park, H. Mustafa, J. Kim, A. Olgun, A. Gollwitzer, D. Senol Cali, C. Firtina, H. Mao, N. Almadhoun Alserr *et al.*, "Genstore: a high-performance in-storage processing system for genome sequence analysis," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022.

[46] K. K. Matam, G. Koo, H. Zha, H.-W. Tseng, and M. Annavaram, "Graphssd: graph semantics aware ssd," in *Proceedings of the 46th international symposium on computer architecture*, 2019.

[47] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, H. Jin, T. Chen, and Z. Jia, "Towards efficient generative large language model serving: A survey from algorithms to systems," *arXiv preprint arXiv:2312.15234*, 2023.

[48] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocessors and Microsystems*, vol. 67, pp. 28–41, 2019.

[49] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," 2019.

[50] OpenAI, "What are tokens and how to count them," https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them.

[51] D. Pandiyan and C.-J. Wu, "Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, 2014.

[52] H. Peng, S. Huang, T. Geng, A. Li, W. Jiang, H. Liu, S. Wang, and C. Ding, "Accelerating transformer-based deep learning models on fpgas using column balanced block pruning," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021.

[53] P. Qi, Y. Song, H. Peng, S. Huang, Q. Zhuge, and E. H.-M. Sha, "Accommodating transformer onto fpga: Coupling the balanced model compression and fpga-implementation optimization," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021.

[54] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[55] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[56] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature nanotechnology*, vol. 15, no. 7, pp. 529–544, 2020.

[57] Y. Seo, J. Choi, S. Cho, H. Han, W. Kim, G. Ryu, J. Ahn, Y. Cho, S. Choi, S. Lee *et al.*, "13.8 a 1a-nm 1.05 v 10.5 gb/s/pin 16gb lpddr5 turbo dram with wck correction strategy, a voltage-offset-calibrated receiver and parasitic capacitance reduction," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, 2024.

[58] G. Shen, J. Zhao, Q. Chen, J. Leng, C. Li, and M. Guo, "Salo: an efficient spatial accelerator enabling hybrid sparse attention mechanisms for long sequences," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[59] Y. Sheng, S. Cao, D. Li, C. Hooper, N. Lee, S. Yang, C. Chou, B. Zhu, L. Zheng, K. Keutzer *et al.*, "S-lora: Serving thousands of concurrent lora adapters," *arXiv preprint arXiv:2311.03285*, 2023.

[60] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Re, I. Stoica, and C. Zhang, "FlexGen: High-throughput generative inference of large language models with a single GPU," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[61] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[62] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," *arXiv preprint arXiv:2312.12456*, 2023.

[63] M. team, "MLC-LLM," 2023. [Online]. Available: https://github.com/mlc-ai/mlc-llm

[64] M. Torabzadehkashi, S. Rezaei, A. Heydarigorji, H. Bobarshad, V. Alves, and N. Bagherzadeh, "Catalina: In-storage processing acceleration for scalable big data analytics," in *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2019.

[65] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[66] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, 2017.

[67] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[68] Y. Wang, X. Pan, Y. An, J. Zhang, and G. Reinman, "Beacongnn: Large-scale gnn acceleration with out-of-order streaming in-storage computing," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 330–344.

[69] M. Wilkening, U. Gupta, S. Hsia, C. Trippel, C.-J. Wu, D. Brooks, and G.-Y. Wei, "Recssd: near data processing for solid state drive based recommendation inference," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021.

[70] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *Proceedings of the 40th International Conference on Machine Learning*, 2023.

[71] W. Xu, J. Kang, and T. Rosing, "A near-storage framework for boosted data preprocessing of mass spectrum clustering," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[72] Y. Xu, W. Li, P. Vaezipoor, S. Sanner, and E. B. Khalil, "Llms and the abstraction and reasoning corpus: Successes, failures, and the importance of object-based representations," *arXiv preprint arXiv:2305.18354*, 2023.

[73] W. Ye, X. Zhou, J. Zhou, C. Chen, and K. Li, "Accelerating attention mechanism on fpgas based on efficient reconfigurable systolic array," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 6, pp. 1–22, 2023.

[74] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022.

[75] Z. Yuan, L. Niu, J. Liu, W. Liu, X. Wang, Y. Shang, G. Sun, Q. Wu, J. Wu, and B. Wu, "Rptq: Reorder-based post-training quantization for large language models," *arXiv preprint arXiv:2304.01089*, 2023.

[76] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed, "Big bird: Transformers for longer sequences," in *Advances in Neural Information Processing Systems*, 2020.

[77] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?" *arXiv preprint arXiv:1905.07830*, 2019.

[78] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[79] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on fpga devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[80] X. Zhao, S. Yang, K. Xie, Y. Feng, Q. Wang, P. Sang, X. Zhan, J. Wu, and J. Chen, "Error bits recovering in 3d NAND flash memory: A novel state-shift re-program (SRP) scheme," in *IEEE International Conference on Integrated Circuits, Technologies and Applications, ICTA 2023, Hefei, China, October 27-29, 2023*, 2023.

[81] C. Zou and A. A. Chien, "Assasin: Architecture support for stream computing to accelerate computational storage," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022.