



Latest updates: <https://dl.acm.org/doi/10.1145/3695053.3731073>

RESEARCH-ARTICLE

AiF: Accelerating On-Device LLM Inference Using In-Flash Processing

JAEYONG LEE, Seoul National University, Seoul, South Korea

HYEUNJOO KIM, Seoul National University, Seoul, South Korea

SANGHUN OH, Seoul National University, Seoul, South Korea

MYOUNGJUN CHUN, Soongsil University, Seoul, South Korea

MYUNGSUK KIM, Kyungpook National University (KNU), Daegu, South Korea

JIHONG KIM, Seoul National University, Seoul, South Korea

Open Access Support provided by:

Soongsil University

Seoul National University

Kyungpook National University (KNU)



PDF Download
3695053.3731073.pdf
07 February 2026
Total Citations: 0
Total Downloads: 12768

Published: 21 June 2025

Citation in BibTeX format

ISCA '25: Proceedings of the 52nd Annual International Symposium on Computer Architecture

June 21 - 25, 2025
Tokyo, Japan

Conference Sponsors:
SIGARCH

AiF: Accelerating On-Device LLM Inference Using In-Flash Processing

Jaeyong Lee

Seoul National University
Seoul, Republic of Korea
jylee@davinci.snu.ac.kr

Myoungjun Chun

Soongsil University
Seoul, Republic of Korea
mjchun@davinci.snu.ac.kr

Hyeunjoo Kim

Seoul National University
Seoul, Republic of Korea
hjkim@davinci.snu.ac.kr

Sanghun Oh

Seoul National University
Seoul, Republic of Korea
shoh@davinci.snu.ac.kr

Myungsuk Kim

Kyungpook National University
Daegu, Republic of Korea
ms.kim@knu.ac.kr

Jihong Kim

Seoul National University
Seoul, Republic of Korea
jihong@davinci.snu.ac.kr

Abstract

While large language models (LLMs) achieve remarkable performance across diverse application domains, their substantial memory demands present challenges, especially on personal devices with limited DRAM capacity. Recent LLM inference engines have introduced SSD offloading for model parameters to reduce memory footprint. However, the highly memory-bound nature of on-device LLMs makes inference speed heavily dependent on read bandwidth, leading to significant performance degradation due to the limited bandwidth of SSDs. In this paper, we propose an in-flash processing solution for on-device LLM, called Accelerator-in-Flash (AiF), which integrates matrix-vector multiplication (GEMV) operations directly into flash chips. By enabling in-flash GEMV operations, AiF leverages the high internal bandwidth of flash chips without being constrained by the limited external bandwidth. Building on this core structure, AiF employs two novel flash read techniques that were specifically optimized for reading LLM parameters stored in flash memory. AiF achieves a 4x boost in internal read bandwidth during inference with minimal implementation overhead, thanks to its streamlined error correction process. Evaluations on eight real-world LLMs reveal that AiF provides a 14.6x throughput improvement compared to baseline SSD offloading schemes. Furthermore, AiF surpasses in-memory inference, delivering 1.4x higher throughput with a significantly reduced memory footprint.

CCS Concepts

- Hardware → Memory and dense storage; • Computer system organizations → Heterogeneous systems.

Keywords

Flash Memory, In-Flash Processing, On-Device LLM Inference

ACM Reference Format:

Jaeyong Lee, Hyeunjoo Kim, Sanghun Oh, Myoungjun Chun, Myungsuk Kim, and Jihong Kim. 2025. AiF: Accelerating On-Device LLM Inference Using In-Flash Processing. In *Proceedings of the 52nd Annual International*



This work is licensed under a Creative Commons Attribution 4.0 International License.
ISCA '25, Tokyo, Japan

© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1261-6/25/06
<https://doi.org/10.1145/3695053.3731073>

Symposium on Computer Architecture (ISCA '25), June 21–25, 2025, Tokyo, Japan. ACM, New York, NY, USA, 15 pages. <https://doi.org/10.1145/3695053.3731073>

1 Introduction

Deploying large language models (LLMs) directly on edge devices is gaining significant traction. On-device inference eliminates the need to transmit data to cloud servers, drastically reducing network latency and enabling faster response times—an essential advantage for real-time applications such as virtual assistants, chatbots, and autonomous vehicles. Additionally, on-device LLMs reduce operational costs by minimizing reliance on centralized cloud infrastructure, making them a more efficient and cost-effective solution. Keeping sensitive data locally also enhances privacy by reducing the risk of unauthorized access or exposure. These benefits collectively position on-device LLMs as a compelling choice for a wide range of edge computing applications.

While on-device LLMs offer clear advantages, their substantial memory requirements pose a critical challenge to widespread adoption and scalability. In general, LLMs require billions of parameters to achieve sophisticated capabilities, such as deep language understanding, complex reasoning, and coherent, context-aware generation. Consequently, these models often necessitate significant memory, ranging from a few GBs to hundreds of GBs. For example, popular models like Meta's LLaMA-3 series [12] require 8–100 billion (B) parameters, and even the smallest 8B model can require more than 16-GB of memory space to run. In contrast, high-end smartphones typically have only 8 to 12-GB of memory, while most laptops provide around 16-GB. These memory constraints are largely dictated by factors such as battery life, cost, and the compact form factor of edge devices, where DRAM is soldered directly to the mainboard.

One approach to alleviating memory capacity constraints is to offload model parameters to a flash-based SSD, which offers orders of magnitude more capacity than DRAM. However, on-device LLM inference is characterized by high read intensity and low arithmetic intensity (e.g., only 1–2 operations per byte). This makes the lower read bandwidth of SSDs (e.g., 4–8 GB/s) compared to DRAM (e.g., 80–100 GB/s) a critical bottleneck, leading to significant performance degradation in inference speed. To mitigate the overhead of data movement from SSDs, in-storage processing (ISP) has been proposed as a more efficient alternative by integrating accelerators

within a SSD controller [10, 18, 22, 34, 40, 43, 48, 59, 61, 62, 75, 79, 83, 91]. However, existing ISP solutions fail to meet the high read bandwidth demands critical for on-device LLM inference. For instance, real-time applications like chatbots require token generation rates of at least 3 tokens per second [5, 23, 32] to ensure a seamless user experience. Achieving this level of performance for models with more than 30B parameters necessitates read bandwidths nearing 100 GB/s. However, our analysis reveals that existing ISP solutions deliver a maximum bandwidth of only 12.8–19.2 GB/s, which falls far short of the requirements for efficient on-device LLM inference. The key limitation of ISP solutions is that data read from flash chips must traverse *bandwidth-constrained flash channels* to reach the SSD controller, resulting in a significant bottleneck.

In this paper, we investigate in-flash processing (IFP) solutions, that integrate matrix-vector multiplication (GEMV) operations directly into flash chips, as a feasible approach for supporting on-device LLM inference. Since the majority of operations in on-device LLM inference involve matrix-vector multiplications between model matrices and token embedding vectors, in-flash GEMV accelerators eliminate the need to transfer model parameters out of flash memory chips. Furthermore, IFP can leverage the high *internal* read bandwidth of multiple flash chips that operate in parallel. Since the aggregated internal bandwidth from multiple flash chips is significantly higher than the constrained flash channel bandwidth, IFP solutions have a potential to achieve substantial performance gains over ISP solutions.

While prior works [16, 24, 38, 45, 77, 78] have demonstrated the effectiveness of IFP solutions for various data-intensive applications, these solutions are not well-suited for on-device LLM. First, they still cannot meet the high read bandwidth requirements essential for on-device LLM inference. For example, when 16 flash chips in a 1-TB SSD can be read in parallel, their combined read bandwidth provides only up to 25.6 GB/s, which falls significantly short of the read bandwidth required for on-device LLM inference. Second, on-device LLM inference is highly sensitive to errors. Unlike the error-resilient applications targeted by prior IFP studies [24, 38, 45], even minimal error rates can cause a significant drop in LLM inference accuracy. This underscores the necessity of incorporating robust on-chip error correction codes (ECC). However, designing a high-throughput on-chip ECC decoder capable of handling the high error rates of flash memory while meeting the substantial bandwidth requirements of LLM inference presents a considerable challenge.

To address the key limitations of existing IFP solutions, we propose a novel IFP solution, Accelerator-in Flash (AiF), specifically designed to optimize in-flash LLM inference. We refer to flash chips with AiF support as AiFChips, and SSDs built with these chips as AiFSSDs. Recognizing that the primary challenges of existing IFP solutions for in-flash LLM inference stem from flash read operations, our approach centers on enhancing the conventional flash read procedure to ensure it operates in a *fast* and *reliable* fashion. To satisfy a high read-bandwidth requirement, we devise a new read command, *charge-recycling* read (denoted by cr-read) which skips time-consuming two steps (i.e., the precharge and discharge steps) of the conventional flash read procedure. When successive wordlines in a block are read, we observed that most wordlines in the block can reuse existing voltage settings between successive reads without the precharge and discharge steps. A large volume

of model parameters, which were sequentially allocated within the same flash block, can be efficiently accessed using cr-reads, achieving 2.8x increase in the read bandwidth.

To support reliable in-flash reads, we take a 2-step approach. First, we devised a new state encoding scheme, *bias-error encoding* (denoted by be-enc), for TLC flash memory that favors LSB pages over CSB and MSB pages in data reliability. The proposed be-enc scheme, which can be implemented with no additional circuit, reduces bit-error rates of LSB page by 87.5% over the other pages. Second, taking advantage of this biased reliability characteristic, AiFSSD strategically stores model parameters only on LSB pages. Thanks to the enhanced reliability of LSB pages, AiFChip can fully recover data inside a flash chip with a highly compact on-chip ECC decoder dedicated to LSB pages only, while maintaining minimal power, performance, and area (PPA) overhead.

By combining the two proposed techniques, AiFChip improves the read bandwidth by 4x while reducing the energy consumption of a read operation by 72.1%. We validated the improvements of AiFChip over a normal flash chip through extensive real device characterization and analog and mixed-signal circuit simulations using industry-standard EDA tools [6, 7, 74]. In a 1-TB AiFSSD, which consists of 16 AiFChips, the internal read bandwidth reaches 102.4 GB/s during LLM inference with minimal impact on area and power consumption.

To evaluate the effectiveness of the AiF at the system level, we developed a full system AiF simulator by integrating the state-of-the-art SSD emulator, NVMeVirt [41], with the widely used on-device LLM inference engine, llama.cpp [17]. Our evaluation results using eight real-world LLMs show that AiF provides a token generation throughput of 5.74 tokens/s for 20B models (2.7 tokens/s for 40B models), delivering over 14.6x the performance of baseline SSD offloading without IFP. Furthermore, compared to the in-memory inference, which stores the entire model in device memory, AiF achieves 1.4x higher throughput while significantly reducing memory footprint. These notable results demonstrate that AiF effectively addresses the limitations of existing solutions and offers a promising approach for efficient, cost-effective on-device LLM deployment.

2 Background

2.1 NAND Flash Organization

Figure 1 shows the organization of a 3D NAND flash-based SSD. A flash cell is the smallest unit for storing data, holding bit information based on its threshold voltage (V_{TH}) level, which largely depends on the charge in its charge trap layer. It means that different bit data can be encoded by different V_{TH} of the cell. For example, we can assign data value ‘0’ to a higher V_{TH} and ‘1’ to a lower V_{TH} . A set of vertically connected cells forms a NAND string. Each string connects to a bitline (BL), and multiple strings on different BLs form a block. Cells at the same vertical location share a control gate connected to a common wordline (WL), enabling all cells on that WL to operate concurrently. Within each plane, all blocks share the BLs. Each BL is connected to a sense amplifier (not shown in Figure 1) in the peripheral circuitry, which detects the values stored in the cells during read operations. Each flash chip, containing multiple planes (typically four), communicates with the controller through a dedicated flash channel.

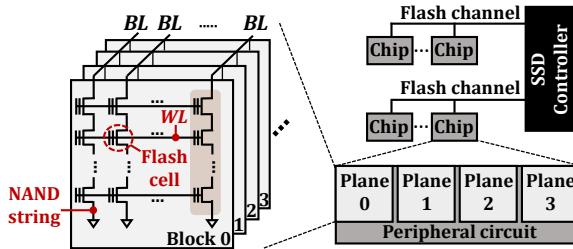


Figure 1: An organizational overview of an SSD.

2.2 NAND Flash Operation

There are three basic operations that enable access to flash memory: program, erase, and read. To increase the V_{TH} of a flash cell, the program operation injects electrons into the charge trap layer of the flash cell by applying a high voltage (e.g., > 20 V) to the WL. On the contrary, the erase operation applies a high voltage (> 20 V) to the substrate (i.e., channel) to remove electrons from the charge trap layer, which decreases the V_{TH} of all flash cells in a block.

During a read operation, NAND flash memory determines a cell's data (i.e., the cell's V_{TH} level) by identifying whether current flows through the corresponding BL. Figure 2 depicts the read mechanism of NAND flash memory that consists of three phases: 1) precharge, 2) evaluation, and 3) discharge.

In the precharge phase (1) in Figure 2, the WLs and BLs are charged to the specific voltages needed to read the values stored on the target WL. A read reference voltage (V_{REF}) is applied to the target WL in the block, while a pass-through voltage (V_{PASS}) is applied to the non-target WLs. Simultaneously, the BLs are precharged in preparation for data sensing. During the evaluation phase (2), the stored values in the target cells are detected. If the (V_{TH}) of the target cell is below V_{REF} , the cell turns on, allowing current to flow. Conversely, if V_{TH} is higher than V_{REF} , the cell remains off, blocking the current. The sense amplifier connected to each BL determines the V_{TH} of the target cell by detecting the current sinking from the BL capacitor (C_{BL}). Due to the high resistance of the BL and NAND string, this sink current is minimal, causing only a slight change in the voltage level of C_{BL} , even when the target cell is on [63]. Finally, in the discharge phase (3), the WLs and BLs of the target block are discharged to return to standby state (0). Discharging the WLs and BLs ensures that subsequent reads, writes, or erases are unaffected by the previously accessed block [19, 63, 68].

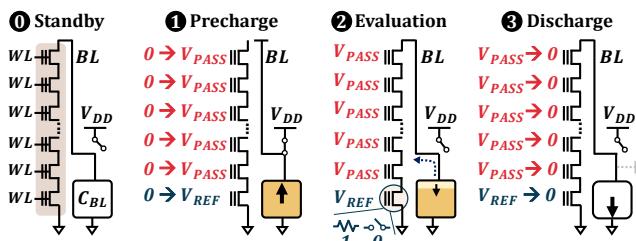
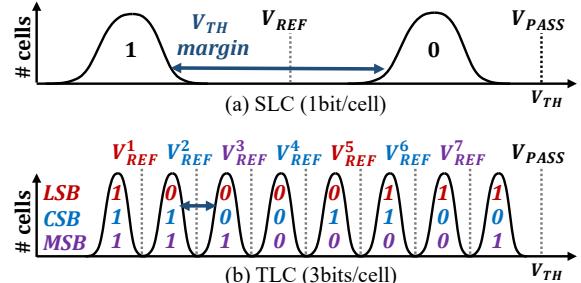


Figure 2: A read sequence of NAND flash memory.

Figure 3: V_{TH} distributions of NAND flash memory.

2.3 Multi-level Cell Technology

To increase the storage capacity, a flash cell can store multiple bits by adjusting its V_{TH} level more precisely, called *m-bit multi-level cell* technology. It stores m bits within a single flash cell by using 2^m distinct (V_{TH}) states (i.e., m is 2 and 3 for MLC and TLC flash memory, respectively). Figure 3 compares the V_{TH} distribution of a SLC with the V_{TH} distribution of a TLC that uses different 2^3 states to store 3 bits per cell, indicating each WL contains three pages called LSB, CSB, and MSB page.

While multi-level cell technology offers high storage capacity, it also brings performance degradation and reliability concerns. SLC NAND, for example, requires only a single sensing operation per read, enabling faster access times. In contrast, TLC NAND requires multiple sensing operations to read a page, which increases read latency; for instance, reading a CSB page in TLC NAND demands three sensing operations (V^2_{REF} , V^4_{REF} , and V^6_{REF}). Furthermore, NAND flash is inherently prone to errors due to its imperfect physical characteristics. Over retention time after programming, cells gradually leak stored charge, causing shifts in the V_{TH} . These shifts can lead to misjudgment of the V_{TH} state during sensing, resulting in read errors. SLC NAND benefits from a large V_{TH} margin between states, contributing to its low error rate. In contrast, TLC NAND has a narrower V_{TH} margin, making it more susceptible to errors. Moreover, as cells undergo more program and erase (P/E) cycles, they gradually wear out, making charge leakage more likely and leading to increased errors. As a result, modern SSD controllers employ complex ECC engines to ensure data integrity before transferring the data to the host.

3 Motivation

3.1 Challenges of On-Device LLM Inference

While LLMs offer powerful capabilities, their massive memory requirements pose a significant barrier to deploying them on resource-constrained personal devices. According to the LLM scaling law [36], as models get larger, they not only improve generation accuracy but also exhibit emergent abilities such as multi-step reasoning. However, this increase in parameters, often reaching billions or more, comes with considerable memory requirements, creating a major bottleneck for on-device LLM inference. Although model lightweighting techniques like quantization [9, 15, 20, 53, 81, 88] can reduce model size, they do not solve the fundamental scalability challenges. For example, models with more than 16B parameters

require over 8-GB of memory, even with aggressive INT4 quantization.

One natural way to alleviate memory constraints is to offload models to SSDs. As secondary storage, SSDs offer orders of magnitude more capacity than DRAM, due to their 50x higher bit density and advanced multi-chip packaging technology [66]. Leveraging this additional storage, the host can perform inference for larger models by dynamically loading parameters into memory on demand.

While SSDs offer large capacity, their limited read bandwidth (4-8 GB/s) becomes a critical performance bottleneck [3, 21, 82, 87]. On-device LLM inference exhibits extremely low arithmetic intensity [3, 21] (only 1–2 operations per byte), primarily due to their single-batch processing characteristic, making inference speed highly dependent on the read bandwidth of model parameters. Additionally, the autoregressive nature of LLM inference necessitates reading the entire model parameters for each token generation step [3, 21, 71]. This dependency places an upper bound on the token generation rate, expressed in tokens per second (tokens/s), as follows:

$$\text{max. tokens/s} \leq \frac{\text{read bandwidth (GB/s)}}{\text{model size (GB)}}. \quad (1)$$

This constraint implies that, with 40 GB of parameters (e.g., 40B model [4]) offloaded to an SSD, the maximum token generation rate is theoretically limited to less than 0.1–0.2 tokens/s. From equation 1, it becomes evident that supporting on-device LLMs using an SSD as the storage medium requires increasing the SSD's read bandwidth by at least two orders of magnitude.

3.2 Existing Strategies for Limited Bandwidth

3.2.1 In-Storage Processing Approach. To address the limited external PCIe bandwidth of SSDs, numerous studies have proposed ISP solutions [10, 18, 22, 34, 40, 43, 48, 59, 61, 62, 75, 79, 83, 91], which minimize data movement by processing data at the SSD controller and delivering only the results to the host. A key advantage of ISP is its ability to leverage existing resources in the SSD controller, such as embedded processors and ECC engines, making it a cost-effective solution. However, existing ISP solutions cannot meet the high read bandwidth demands essential for on-device LLM inference. In ISP solutions, data read from flash chips must be transferred to the SSD controller via flash channels, which use a multi-drop bus topology. This topology limits data transfers to one flash chip per channel at a time, creating a significant bottleneck at the flash channels. With consumer-grade SSDs typically offering up to 8 channels [25, 64, 72], each limited to 1.6–2.4 GB/s of bandwidth, ISP solutions can only achieve an aggregate bandwidth of 12.8–19.2 GB/s. In contrast, on-device LLM inference often requires read bandwidths nearing 100 GB/s, rendering ISP solutions inadequate for such demanding requirements.

3.2.2 In-Flash Processing Approach. Another strategy to increase the read bandwidth of SSDs for LLM inference is to employ SSDs with in-flash processing capabilities. Modern SSDs are composed of multiple flash chips (e.g., 16 chips in a 1-TB SSD), whose aggregate read bandwidth—referred to as internal bandwidth—far surpasses the constrained bandwidth of flash channels. By enabling

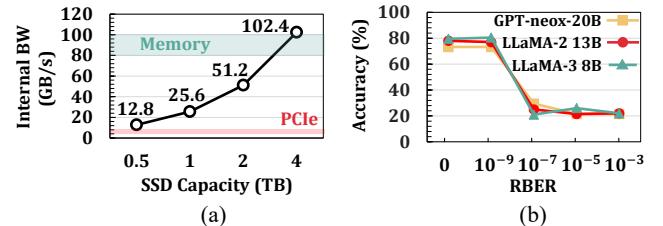


Figure 4: (a) Internal bandwidth of SSDs. (b) Impact of RBER on inference accuracy.

computations to occur directly within the flash chips through dedicated in-flash accelerators, SSDs can effectively leverage their high internal bandwidth to support efficient LLM inference. Furthermore, since LLM inference primarily involves GEMV, which inherently reduces data size by transforming input matrices (e.g., model parameters) into smaller output vectors (i.e., token embedding vectors). This high data reduction characteristic makes on-device LLMs particularly well-suited for IFP. For instance, a large matrix (e.g., 175-MiB) stored across flash chips can be processed using its high internal bandwidth, with only the significantly smaller output vector (e.g., 16-KiB) transferred over the limited flash channel bandwidth.

3.3 Challenges of IFP-based LLM Inference

Prior works [16, 24, 38, 45, 77, 78] have demonstrated the effectiveness of IFP in various data-intensive applications. Furthermore, as discussed in Section 3.2.2, the inherent characteristics of LLM inference shows the significant potential of IFP for LLM inference. However, we identify two key challenges in incorporating IFP for LLM inference that stem from the low-level characteristics of modern SSDs.

Challenge 1: High Read Bandwidth Requirement. Although IFP enables SSDs to fully utilize their internal bandwidth, it still falls significantly short of the bandwidth required to meet the commonly desired throughput for on-device LLM inference. Figure 4(a) shows the internal bandwidth of TLC SSDs. A 1-TB TLC SSD with 16 flash chips [25, 72], a popular configuration in personal devices like laptops and smartphones, provides an internal bandwidth of only 25.6 GB/s. This limitation means that when running inference on a 30B model, a 1-TB SSD only achieves less than 0.85 tokens/s even with IFP.¹

Challenge 2: High Reliability Requirement. Unlike error-resilient applications targeted by prior IFP studies [24, 38, 45], LLM inference is highly vulnerable to errors. To examine the impact of raw bit error rate (RBER) on LLM inference, we evaluated how the model accuracy changes as the number of error bits in model parameters increases by injecting bitflip errors into model matrices.² Figure 4(b) shows the average model accuracy as RBER increases, revealing notable sensitivity to even extremely low RBER ranges. For example, the LLaMA-3 8B model (INT8-quantized) experiences an accuracy drop of over 60% at an RBER of 10^{-7} , which is far below the typical RBER ranges of NAND flash memory ($> 10^{-3}$). This highlights

¹One way to increase internal bandwidth is to scale up SSD capacity by adding more flash chips (e.g., 4-TB capacity). However, this significantly raises the cost of edge devices, undermining the benefit of IFP.

²For accuracy evaluation on HellasSwag [89], all models were quantized to INT8. We then injected RBER-based bit-flips probabilistically, but only in the QKV projection and FFN weight matrices (see Figure 15).

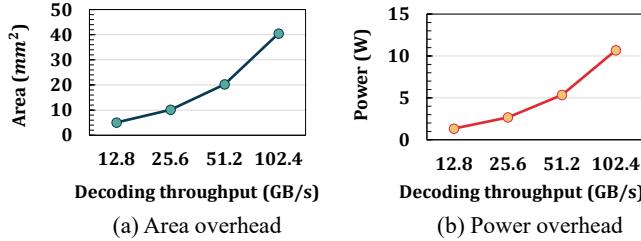


Figure 5: PPA impact of ECC decoder.

that LLM inference is not inherently error-tolerant, making ECC decoder within the flash chip (i.e., on-chip ECC) indispensable to ensure the inference accuracy.

While on-chip ECC is essential, it introduces significant overheads in power, performance, and area (PPA) perspective. This is because on-chip ECC decoders must achieve high decoding throughput while maintaining a robust error correction capability (i.e., high hardware complexity) to accommodate the high RBER of NAND flash memory. For conventional off-chip ECC decoders implemented at the SSD controller, error correction is primarily performed on data delivered to the host via the PCIe interface, resulting in the decoding throughput requirement is dependent on the external bandwidth. In contrast, on-chip ECC decoders perform error correction directly within each flash chip, requiring the total correction throughput across all flash chips to match the SSD's internal bandwidth. Since the overhead of ECC scales linearly with throughput, this imposes a substantial burden on SSDs.

Figure 5 illustrates the estimated area and power consumption of a BCH-based ECC decoder at increasing decoding throughputs. The ECC decoder is configured to correct 50-bit errors per 1-KiB, based on flash error rates from our real device characterizations (Section 4.3) and prior studies [49, 68]. To achieve 102.4 GB/s throughput, the cumulative on-chip ECC decoders demand 40.12 mm^2 of silicon area and consume 10.694 W, which far exceeds the power budget of consumer-grade SSDs (6-8 W [25, 72]). These results emphasize the substantial PPA overhead of high-throughput yet complex on-chip ECC, presenting significant challenges for applying IFP to on-device LLM inference.

4 AiF: Accelerator-in-Flash

4.1 Overview

We propose AiF, a novel IFP solution specifically designed to optimize in-flash LLM inference. Figure 6 presents a structural overview of AiF. A key component of AiF is the AiFSSD, which is composed of multiple GEMV-enabled flash chips, referred to as AiFChips. (For simplicity, only four chips are depicted.) Unlike baseline SSD offloading schemes that rely on time-consuming I/O to load matrices during inference, AiF performs GEMV directly on the AiFChips where the matrices are stored. Each matrix is divided into equal-sized sub-matrices and stored sequentially across the AiFChips. This enables the AiFSSD to fully utilize the high internal bandwidth of the SSD by operating all AiFChips simultaneously. During GEMV operations, the SSD controller aggregates the sub-vectors from each chip, combines them, and sends the resulting vector back to the host. To support on-chip GEMV operations, each AiFChip is equipped

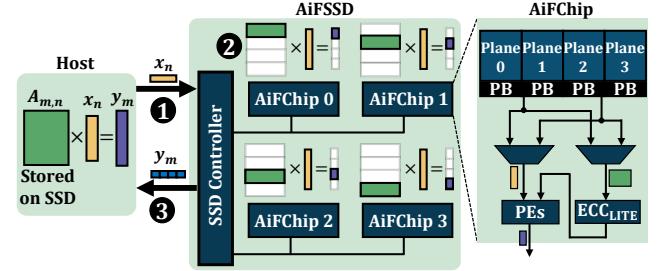


Figure 6: A structural overview of AiF.

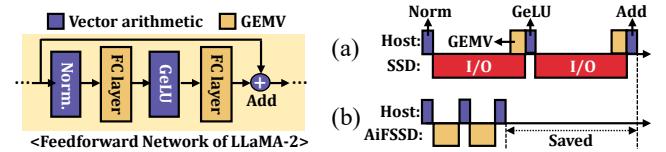


Figure 7: A comparison of (a) the baseline SSD offloading scheme and (b) AiF.

with two components: 1) ECC_{LITE}, a lightweight on-chip ECC decoder, and 2) product elements (PEs), consisting of multipliers and an adder tree for vector products.

Figure 7 illustrates an example of the computational process of AiF by comparing its execution timeline with the baseline SSD offloading scheme during the processing of the feed-forward network in LLaMA-2. In the baseline SSD offloading scheme, the matrix must first be loaded from the SSD into memory for GEMV, creating a significant I/O bottleneck. In contrast, AiF utilizes the SSD's high internal bandwidth to perform GEMV, effectively reducing the GEMV computation time. AiFSSD can be integrated with any LLMs by focusing solely on GEMV, the primary bottleneck in on-device LLM inference, while leaving model-specific vector operations (e.g., activation functions) to the host. This decoupling enhances AiF's flexibility across various models.

To address the key limitations of existing IFP solutions (Section 3.3), which primarily arise from flash read operations, we focus on redesigning the conventional flash read procedure to specifically optimize it for in-flash LLM inference. We introduce two simple yet highly effective techniques, *charge-recycling read* (Section 4.2) and *bias-error encoding* (Section 4.3), that significantly enhance both the bandwidth and reliability of parameter reads during inference. These two techniques combined allow AiF to satisfy the high bandwidth demands of on-device LLM inference (**Challenge 1**) while maintaining inference accuracy through robust error correction, even with the highly compact on-chip ECC_{LITE} (**Challenge 2**).

4.2 Charge-Recycling Read

4.2.1 Key Idea. Cr-read is a technique that specializes the flash read sequence for bulk reads within a block. As described in Section 2.2, conventional flash chips include a discharge phase at the end of each read sequence to return to a standby state. This process ensures that a subsequent flash operation can start from the known initial state of block access. However, we observe that a

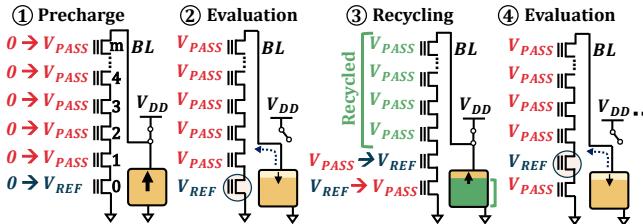


Figure 8: An operational overview of cr-read.

conventional return-to-initial state machine for flash operations may miss optimization opportunities when additional information is available for successive flash operations. Unlike general flash access workloads where subsequent flash operations are difficult to predict, in-flash LLM inference exhibits strong spatial locality when reading model matrices within a flash block. As a result, most voltages applied to WLs and BLs from previous reads can be safely reused without requiring the discharge phase. By leveraging these voltage recycling opportunities inherent in reading successive WLs within a block, cr-read enables AiFChip to read large volumes of LLM parameters much faster.

Figure 8 illustrates how the proposed cr-read scheme works when WL₀ and WL₁ are successively read. In cr-read, after sensing a target WL (② in Figure 8), the WLs and BLs are not fully discharged. Instead, only the next read WL discharges from V_{PASS} to V_{REF}, while the previously read WL charges from V_{REF} to V_{PASS}, with all other WLs holding V_{PASS} for the next read (③). Additionally, the BLs are only slightly recharged to compensate for any voltage drop during the previous evaluation phase. Although cr-read can only be applied when successive reads to the same block are requested without any interval, it significantly enhances both read latency and energy efficiency—key design requirements for on-device LLM inference. Figure 9 compares cr-read with conventional reads when reading n WLs consecutively within a block. As shown in Figure 9(a), conventional reads require time-consuming precharge and discharge phases for each WL. The read latency (tR) for each WL in conventional reads can be expressed as:

$$tR = (tPRE + tEVAL + tDISCH) \quad (2)$$

where tPRE, tEVAL, and tDISCH represent the latencies of the precharge, evaluation, and discharge phases, respectively³. In contrast, as shown in Figure 9(b), cr-read requires only two phases per read: recycling and evaluation (except for the first and last WL reads). Therefore, in cr-read, tR is given by:

$$tR = (tRECY + tEVAL) \quad (3)$$

where tRECY denotes the latency of the recycling phase. Moreover, tRECY is considerably shorter than tPRE because tPRE requires charging *all* WLs from 0 V to V_{PASS} (6 V) (i.e., full-range voltage swing), resulting in a high RC delay. In contrast, tRECY only needs to charge *a single* WL from V_{REF} (3.5 V) to V_{PASS} (6 V). The cr-read scheme is effective as well in reducing the energy consumption during read operations. Unlike conventional reads, cr-read only

³In this section, we assume that the WLs operate as SLC NAND, requiring only single sensing. This is because applying the bias-error encoding technique, which will be detailed in Section 4.3, allows all LLM parameters to be read as if they were SLC.

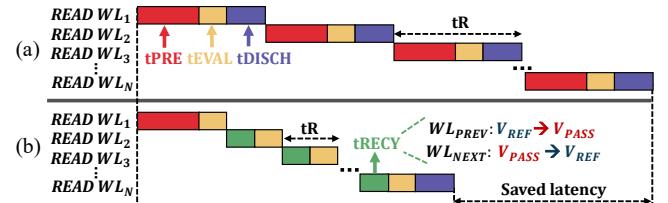


Figure 9: A comparison of (a) conventional read sequence and (b) cr-read.

partially charges the previously read WL, raising it from V_{REF} to V_{PASS}. In addition, since the voltage drop across the BLs during evaluation is minimal, only a small amount of current is required to recharge the BLs for sensing the next WL. Consequently, cr-read substantially reduces energy consumption during read operations.

To maximize the bandwidth and energy efficiency of AiFChips during inference, AiFSSD arranges LLM parameters in a layout optimized for cr-read. Since cr-read achieves peak efficiency when multiple WLs within the same block are read consecutively, AiFSSD organizes each matrix within a same block, allowing for bulk reading of the entire matrix through consecutive cr-read operations. Since LLM parameters have *write once read many* characteristics, AiFSSD can statically pre-arrange each matrix within the same block. Once a matrix is organized within a block, cr-read can be applied for all inference requests to a given model.

4.2.2 Evaluation of cr-read. To validate the effectiveness and practical feasibility of the proposed cr-read scheme, we conducted various characterization studies using both SPICE circuit-level simulations⁴ and real NAND flash cell array⁵. To accurately represent the flash cell model in our simulations, we fine-tuned the cell model based on actual measurements from a fabricated real flash cell array. To ensure the accuracy of our simulations, we compared the tR and power obtained from our simulator with the measured tR (*single sensing read*) of 28 us from the actual product and a reported power value of 24.22 mW [19, 30]. The differences in tR and power consumptions were 2.9% and 0.9%, respectively.

Read Latency Validation. Figure 10(a) compares the read latency (tR) of conventional reads and cr-read. By replacing the time-consuming precharge and discharge phases with an efficient recycling phase, which introduces only a 6.04 μ s delay due to reduced capacitive loads and voltage swing range, cr-read reduces

⁴Following established methodologies [65, 70, 76], we first conducted measurements on real charge trap flash (CTF) cells, which are widely used for 3D flash memory devices. We focused on the I_d - V_g (drain current vs. gate voltage) characteristics. These provide a foundation for an accurate flash cell model that reflects actual cell behavior. Second, based on the data collected from measurements, we fine-tuned the cell model using the BSIM [1, 11]. At this stage, we verified the model's reliability by comparing the I_d - V_g curves of the tuned BSIM model to those of the real CTF cells, observing an 8.02% difference in drain current, which falls within an acceptable range. Third, circuit modeling. We leveraged the tuned BSIM models and Cadence's Virtuoso [7] to design the NAND flash cell array. The required WL and BL capacitance values were obtained from previous research [35, 42, 60]. Lastly, for simulation, we used Cadence's Spectre [6], an industry-standard SPICE simulator for analog and mixed-signal circuits.

⁵Although simulations offer precise estimations, unexpected real-world factors may impact cr-read's correct operation on actual flash chips. To ensure the functionality of cr-read, we conducted additional testing on a fabricated CTF cell array chip. This array, a 9 (WLs) by 9 (BLs) cell configuration, allows fine-grained control over each WL and BL, enabling us to validate cr-read's behavior in a realistic environment.

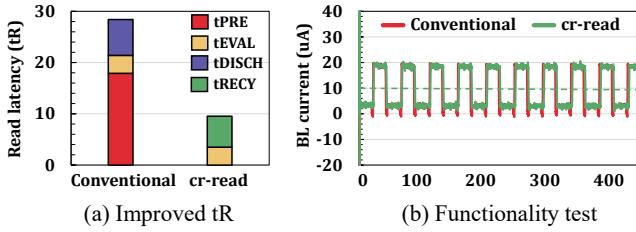


Figure 10: Evaluation results of cr-read.

tR by 64%. This substantial reduction in tR allows the AiFChip to achieve an effective bandwidth of 6.4 GB/s when consecutive cr-reads are applied within a block, representing a 2.8x improvement over conventional flash chips.

Energy Efficiency Validation. By minimizing the repetitive precharge of WLs and BLs, cr-read significantly improves the energy efficiency of flash reads. Simulation results indicate that cr-read reduces read energy consumption by 72.1%, from 18.278 pJ/bit to 5.098 pJ/bit.

Functionality Validation. To validate the real-world functionality of cr-read, we perform experiments using a fabricated CTF cell array. In these experiments, we read the same set of pages using both conventional reads and cr-reads, then compare the resulting data to determine whether cr-read introduces any data distortion. Since the tested chip lacks dedicated sense amplifiers, we indirectly assess the data by measuring the current flowing through the BLs during the evaluation phase. As illustrated in Figure 10(b), the measured BL current during cr-read matches that of conventional reads (i.e., produces identical results), confirming that cr-read does not alter the read data and functions correctly.

4.2.3 Feasibility. Another key advantage of cr-read is its ease of implementation. Implementing cr-read requires two main functions: voltage control for each WL and timing control for precharge and discharge phases, both already supported by flash chips. Voltage control can be managed through the X-decoder [37], which selects and drives the appropriate WLs with the required voltages, while timing control can be adjusted by modifying the timer code in the flash scheduler [80]. These adjustments involve only minor modifications to the existing flash chip logic.

4.3 Bias-Error Encoding

4.3.1 Key Idea. Be-enc effectively reduces bit error rates in the stored LLM model while slightly sacrificing both read latency and bit error rates for read operations to general pages, i.e., pages that do not store the LLM model. This design is based on two key observations: 1) increased tR for general pages has minimal impact on system-level performance, and 2) modern SSDs have a large ECC-capability margin.

Observation 1: tR Tolerance. Figure 11(a) shows the internal bandwidth of a 1-TB SSD as read latency (tR) is incrementally increased from 40 μs to 80 μs. Despite the increase in tR, the internal bandwidth remains constrained by the external bandwidth (e.g., 4-8 GB/s). This indicates that even as tR rises (and flash chip bandwidth decreases), the effective user-perceived read bandwidth of the SSD is minimally impacted. From a latency perspective, increases in tR also have limited impact. This is because end-to-end latency

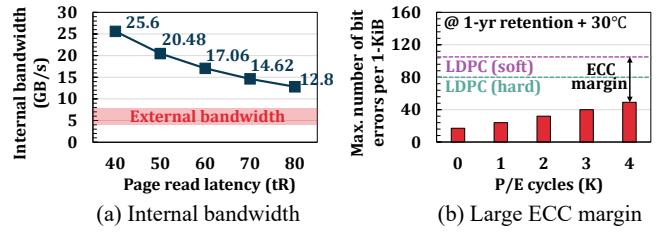


Figure 11: Characteristics of modern SSDs: (a) Variations of internal bandwidth over varying tR and (b) Changes in bit errors over increasing P/E cycles.

includes more than just page read latency; it encompasses data traversal through multiple layers, such as the flash channel, PCIe interface, and kernel I/O stack. As a result, slight increases in tR can be amortized within the overall latency, exerting limited effect on perceived performance.

Observation 2: Large ECC margin. As previous studies have shown, the error correction capability of the ECC engine employed in modern SSDs significantly surpasses the actual RBER of flash pages, resulting in a large ECC margin [8, 49, 68]. Figure 11(b) shows the maximum number of bit errors per 1-KiB data across various P/E cycles, based on our characterization of 160 real TLC flash chips⁶. We also plot the correction capabilities for both hard and soft decision decoding in LDPC-based ECC decoders commonly used in SSDs. As shown in the figure, a significant ECC margin exists even under the worst-case operating conditions prescribed by manufacturers (e.g., a 1-year retention time at 4K P/E cycles). This substantial margin stems from two factors: the significant portion of ECC parity reserved in modern SSDs (e.g., 2 KB of parity per 16-KiB data [49]) and the robust error correction capabilities provided by LDPC soft-decision decoding [44, 52, 90].

Based on these observations, the proposed be-enc scheme intelligently manages tR and data reliability for the three pages stored within a single WL of TLC NAND flash memory. Be-enc intentionally introduces heterogeneity in the reliability and performance between pages, storing IFP data (i.e., LLM parameters) only on the most reliable and fastest pages. By prioritizing the pages containing IFP data, be-enc effectively increases the SSD's internal read bandwidth during inference while simplifying the on-chip ECC requirements. Be-enc is achieved through the reconfiguration of the V_{TH} state encoding in flash memory. It is important to note that prioritizing the reliability of IFP data over general I/O data ensures that IFP data can be reliably decoded within a flash chip using a compact on-chip ECC decoder. Furthermore, **Observations 1 and 2** (above) indicate that general I/O data does not experience significant degradation in performance or reliability as a result.

Figure 12 illustrates how the prioritized be-enc scheme differs from the conventional V_{TH} state encoding. Figure 12(a) shows the conventional V_{TH} state encoding used in TLC NAND, where (x, y, z) coding represents the number of sensing operations required

⁶We characterized 160 real 3D TLC flash chips using an FPGA-based testing platform equipped with a custom flash controller and a temperature-controlled environment. To ensure comprehensive testing, we selected 120 blocks from each chip, evenly distributed across various physical locations, and evaluated all wordlines (WLs) within each block, totaling 3,686,400 WLs (11,059,200 pages). This extensive testing setup enabled us to obtain statistically significant experimental results. All test procedures follow the JEDEC standards common in the memory industry.

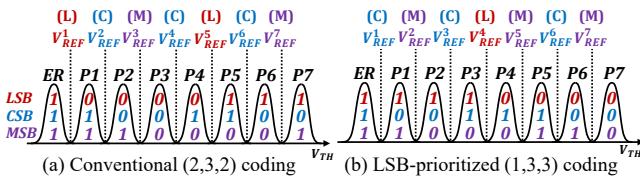


Figure 12: A comparison of (a) conventional V_{TH} state encoding and (b) proposed V_{TH} state encoding.

for reading each page type (i.e., LSB, CSB and MSB). In the (2,3,2) coding, for example, the LSB page requires two sensing operations, while the CSB page requires three. Modern SSDs typically use (2,3,2) coding to balance read time (t_R) across all pages. Figure 12(b) shows the proposed LSB-prioritized (1,3,3) coding. By reconfiguring the state encoding, the LSB page requires only one sensing operation, making its read performance equivalent to SLC, while the sensing count for the MSB page increases from two to three. Leveraging this intentional t_R variation, AiFSSD strategically stores IFP data only on the LSB pages. As a result, during inference, all parameters are read solely from the fastest LSB pages.

Another key advantage of be-enc is the enhanced reliability of the LSB pages, which store IFP data. Figures 13(a) and 13(b) present the maximum number of bit errors per 1-KiB data for each page type under (2,3,2) and (1,3,3) coding, respectively. Compared to (2,3,2) coding, (1,3,3) coding reduces the bit errors of LSB pages by 80%. In (1,3,3) coding, LSB page errors occur only at V_{REF}^4 , which distinguishes the P3 and P4 states. Since these states are in the most stable voltage range [39, 46, 49, 86], where fewer V_{TH} shifts occur, the error rates of the LSB pages remain low even under heavy wear (e.g., 4K P/E cycles) and 1-year retention. This significant reduction in errors allows AiFChip to implement a lightweight on-chip ECC.

4.3.2 Feasibility & Overhead. Existing flash chips already support dynamic reconfiguration of the V_{TH} state encoding at runtime [58, 63]. This capability allows be-enc to be applied without requiring any hardware modifications to the flash chip.

A major drawback of be-enc is the increase in the sensing count for MSB pages from 2 to 3, resulting in a longer t_R for MSB pages. To minimize this overhead, AiFSSD applies be-enc selectively on a per-block basis. Specifically, AiFSSD categorizes flash blocks into two groups: 1) non-IFP blocks, which handle only standard I/O operations and use the conventional (2,3,2) coding, and 2) IFP blocks, which employ (1,3,3) coding and store both LLM parameters (LSB pages) and general data (CSB/MSB pages). IFP blocks are allocated dynamically from available free blocks whenever LLM parameters need to be stored. By applying be-enc selectively to IFP blocks, only

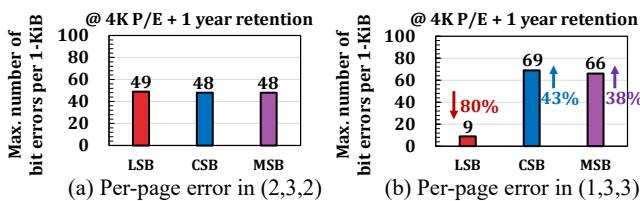


Figure 13: Impact of V_{TH} state encoding on page error rates.

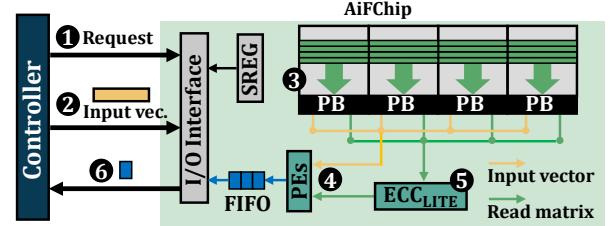


Figure 14: On-chip computation flow.

Table 1: Breakdown of area and power.

Module Name	Area [mm ²]	Avg. Power [mW]
ECC_LITE	0.167	45.1
Product Elements (PEs)	0.026	3.98
Others	0.016	2.6
Total	0.209	51.68

those blocks containing LLM parameters experience performance degradation. We evaluate overhead of be-enc in detail in Section 6.2.

4.4 Design of AiFChip

This section presents the detailed design of AiFChip, highlighting its key architectural features and analyzing its overhead.

Page Buffer Reuse. As illustrated in Figure 6, each AiFChip performs GEMV operations by multiplying a submatrix, stored sequentially in the IFP block, with an input vector received from the host. This input vector, typically between 8-KiB and 32-KiB, is loaded onto the AiFChip to serve as an operand for GEMV computations.

Instead of requiring a dedicated buffer (e.g., SRAM) for loading the input vector, AiFChip reuses the existing page buffer (PB) allocated to each plane. Originally, the PB functions as a temporary buffer for read, program, and data transfer operations and comprises multiple page latches, each sized to hold a single page. In TLC NAND, each PB includes at least four page latches [24, 26, 63, 69]: one serves as a cache buffer for data transfer, while the remaining three are used as data buffers for page reads. For example, a CSB page read requires three page latches to store data across three sensing operations. However, unlike general I/O operations, AiFChip requires only a single page latch during inference. This is because be-enc enables parameter reads with a single sensing operation, similar to SLC reads. As a result, the remaining two page latches can be repurposed as buffers to hold input vectors, eliminating the need for additional dedicated buffers.

On-Chip Computation Flow. Figure 14 illustrates the overall flow of on-chip GEMV computation. The process begins with the controller sending a command to initiate the GEMV operation (1 in Figure 14), specifying key parameters: the input vector's dimension, the ID of the block containing the matrix, the offset within the block, and the number of pages to read. The controller then loads the input vector into the PBs (2), after which planes read matrix sequentially in a continuous flow (3). As matrix segments are read, the AiFChip retrieves them along with the input vector from the PBs, performing multiplication and accumulation in the PEs (4). PEs consist of INT8 multipliers and an adder tree to support GEMV operations. Each matrix segment undergoes error correction through ECC_LITE, a compact and high-throughput on-chip ECC decoder, before computation (5). ECC_LITE comprises a set of BCH

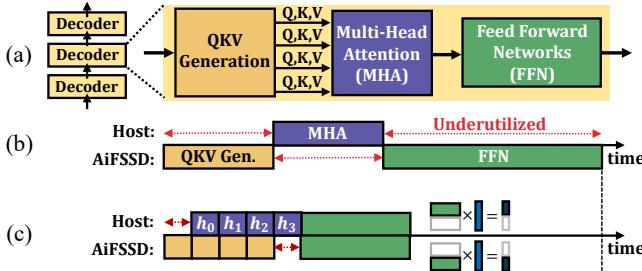


Figure 15: Optimizing LLM inference performance through parallel host-AiFSSD execution: (a) computational flow of LLM inference, (b) a sequential execution flow, and (c) a parallel execution flow.

decoders, corrects up to 10-bit errors per 1-KiB data with a 6.4 GB/s throughput. The correction capability of ECC_{LITE} is based on our characterization results in Figure 13(b). After computations, the resulting output vector elements are stored sequentially in the output FIFO. Throughout the computation, the controller polls each AiFChip in a round-robin manner to retrieve results from the FIFO as they are generated (6), with the FIFO status tracked through the flash chip's status register (SREG).

Area and Power Overhead. For area and power estimation of the AiFChip, we implement the proposed logic in Verilog HDL, synthesizing the designs with Design Compiler at 45 nm technology node and operating frequency of 200 MHz. As detailed in Table 1, the AiFChip's estimated area overhead is 0.209 mm^2 , approximately 0.2% of the total flash chip area. The average power consumption is 51.68 mW, with over 87% attributed to ECC. Compared to the baseline ECC decoder analyzed in Figure 5, ECC_{LITE} requires 15.01× less area and consumes 14.83× less power at the same throughput, enabled by significantly reducing its error correction capability through the use of be-enc.

5 Integrating AiF into On-Device LLM Inference

Designing an efficient on-device LLM inference system with AiFSSD requires substantial multi-layer development from application-level software to system software and down to the AiFSSD firmware. In this section, we outline key considerations for building such systems, aiming to provide a broader perspective on AiF.

5.1 LLM Inference Workflow with AiFSSD

LLM inference consists of two distinct phases: a prefill phase, which processes the user's input prompt in parallel, and a decode phase, which generates tokens sequentially. In a host system with AiFSSD attached, only the decode phase is offloaded to AiFSSD, while the prefill phase is handled by the host. This phase splitting is based on the consideration that the decode phase requires repeated reads of the entire model for each token generation, making it a key bottleneck in inference [3, 21, 82, 87]. By contrast, the prefill phase requires only a single read of the model and can leverage the host's high computing power to process tokens in parallel.

Figure 15(a) illustrates the computational flow of LLM inference during a decode phase. LLM inference progresses through a series of decoder blocks, where each block takes the output vector (i.e.,

token embedding) of the previous block as its input. Each decoder block is composed of three main computational stages: QKV generation, multi-head attention (MHA), and feed-forward networks (FFNs). In the QKV generation stage, the input vector undergoes linear projections to produce multiple sets of Query (Q), Key (K), and Value (V) vectors. In the MHA stage, each set of Q, K, and V vectors is processed along with previously generated K and V vectors (stored in the KV cache in memory) to produce an attention head. Each attention head is generated independently and then aggregated to form a context-aware token embedding. Finally, in the FFN stage, the token embedding undergoes further refinement through additional linear projections before passing to the next decoder block.

Figure 15(b) illustrates the straightforward scheduling approach for AiF. In this scheme, AiF offloads all stages except the MHA stage to the AiFSSD. The host remains responsible for the MHA stage because it requires access to the KV cache, which must be stored in memory due to its frequent updates.⁷ A key limitation of this simple scheduling approach is its sequential computation pattern, alternating between the host and AiFSSD instead of engaging them concurrently, resulting in suboptimal performance.

Figure 15(c) illustrates the execution timeline of the proposed parallel scheduling scheme. To enable the overlapped inference with the host, AiF leverages head-level and tensor-level parallelism, inspired by multi-GPU scheduling techniques [73]. As explained above, each attention head (h_i) in the MHA stage can be computed independently with just a single set of Q, K, and V vectors. This independence allows the host to begin processing each head as soon as a vector set is generated, creating an overlap between the Q, K, V generation and MHA stages. Parallel execution is also applied to the FFN stage by dividing the projection matrix into two submatrices, with the host and AiFSSD each processing one independently before aggregating the results. To enable this, the host selectively retains only the necessary portions of the projection matrix in memory during the prefill phase, preparing for parallel processing of the FFN stage in the subsequent decode phase. The number of submatrices that can be loaded (i.e., the degree of tensor-level parallelism) is determined by the available host memory capacity.

5.2 System Supports for AiFSSD Integration

This section outlines key design considerations across multiple layers for efficiently integrating AiFSSD.

SSD Extensions. In our design, AiFSSD introduces two specialized commands, (i) `aif_post` and (ii) `aif_gemm`, which extend the existing NVMe protocol. The `aif_post` command stores a matrix in AiFSSD with an optimized layout for IFP. It specifies (1) the matrix dimensions, (2) the host memory address of the matrix, and (3) the *sequential* LBA range for storing the matrix. Upon receiving `aif_post`, AiFSSD partitions the matrix into equal-sized submatrices and stores them sequentially in IFP blocks across AiFChips (see Figure 6).⁸ These blocks are dynamically allocated from the free

⁷In on-device LLMs, the KV cache is typically much smaller than model weights due to their low-batch characteristic. Table 3 in Section 6.1 details the weight and maximum KV cache sizes (single-batch) of the models evaluated in this work.

⁸Each submatrix is initially written sequentially to the LSB pages of the IFP block, while CSB/MSB pages accommodate incoming general data afterward [67, 84]. Although the `aif_post` command ensures sequential placement of submatrices, SSD

block list, and AiFSSD maintains a per-block bitmap to differentiate them from non-IFP blocks. The `aif_gemv` command issues a GEMV operation. It specifies (1) the input vector dimension, (2) the input vector's memory address, (3) the matrix's LBA range, and (4) the output vector's memory address. Upon receiving `aif_gemv`, AiFSSD locates the start page addresses of each submatrix and sends the GEMV command, along with the input vector, to AiFChips for computation (steps ① and ② in Figure 14).⁹

System Software Modifications. In our design, host applications (e.g., an LLM inference engine) communicate directly with AiFSSD over the NVMe raw block interface, bypassing the kernel I/O stack. Although this approach increases application complexity, it provides key benefits. First, it minimizes I/O stack modifications. Only the NVMe driver requires changes. With a userspace NVMe library such as SPDK [85], OS-level modifications, which is often a major barrier to practical deployment, can be avoided entirely. Second, it can maintain optimized data layout. Because the application manually manages IFP data, its sequential placement remains intact, unaffected by OS services like defragmentation. Third, it reduces communication delays. Bypassing the kernel I/O stack lowers latency, enabling more efficient GEMV execution.

Application-level Requirements. To ensure correct AiF operation, the host application must meet two key requirements: First, it must track matrix LBAs stored in AiFSSD. The application uses the `aif_post` command to store each matrix in AiFSSD. It then records each matrix's LBA range in a *tensor table*, a data structure file that maps tensor identifiers to their corresponding LBAs. During inference, the application consults this table to issue `aif_gemv` with the correct LBA range. Second, it must isolate IFP data. IFP data must reside in a dedicated LBA space, separate from general file system data. This can be achieved by assigning it to a raw partition or a separate NVMe Namespace [14], created using standard Linux tools like `fdisk` or `nvme-cl` in Linux.

6 System-Level Evaluation

We evaluate the effectiveness of AiF at the system level by integrating a state-of-the-art SSD emulator [41] and an on-device LLM inference engine.

6.1 Evaluation Setup

Methodology. We build a virtual AiFSSD using NVMeVirt [41], an open-source SSD emulator. NVMeVirt supports precise timing models of SSD internal operations, such as flash read latency, and enables full-system evaluation by mimicking a virtual PCIe endpoint device. Leveraging these capabilities, the virtual AiFSSD accurately models the data flow and timing characteristics of AiFSSD. As an on-device LLM inference engine, we employ `llama.cpp` [17], a widely adopted on-device engine that offers flexibility and low-level accessibility.

In order to integrate the virtual AiFSSD with `llama.cpp`, we extend NVMeVirt to support `aif_post` and `aif_gemv` requests, as described in Section 5.2. A modified NVMe device driver in the

internal tasks such as garbage collection (GC) may introduce fragmentation. To maintain the optimized layout, AiFSSD performs page copies while preserving the original LSB page order within the IFP block whenever GC is triggered in an IFP block.

⁹Since each submatrix is stored sequentially in IFP blocks, AiFSSD typically needs to look up the mapping table only once per GEMV command. However, if a submatrix spans two IFP blocks, AiFSSD splits the GEMV command into two separate operations.

Linux kernel enables `llama.cpp` to send requests directly to AiFSSD from userspace via the `ioctl` system call.

During inference, `llama.cpp` issues GEMV requests to AiFSSD whenever a matrix offloaded to the SSD requires GEMV computation. AiFSSD then computes the GEMV delay using its internal timing model, generates an output vector, and transfers it to the host with an interrupt. For efficient evaluation, we design the virtual AiFSSD to simulate the delay and provide the dummy vector instead of performing the actual computation. Upon receiving the dummy output vector, `llama.cpp` continues the inference flow as if it were the real GEMV output. Note that the LLM inference flow is deterministic; the contents of the output vector do not affect performance evaluation.

Evaluated Systems. We evaluate the following systems:

- **In-Memory:** The In-Memory scheme represents an ideal system without memory capacity constraints, employing 128-GB of DDR5 memory (up to 86.4 GB/s bandwidth) to store all model parameters in host memory. In-Memory employs an Intel Core i9-14900KS processor as a computation unit. We exclude dedicated accelerators (e.g., NPUs) for on-device LLM inference, as their low arithmetic intensity does not benefit from accelerators when memory bandwidth is consistent.

- **Memory+SSD:** The Memory+SSD scheme is a baseline scheme that offloads model parameters to a 1-TB SSD due to limited memory capacity. We set the memory capacity to 8-GB. In Memory+SSD, the host memory can store only a portion of the model parameters. Whenever `llama.cpp` requires model parameters offloaded to the SSD, Memory+SSD fetches them into host memory before computation.

- **AiF:** AiF employs a 1-TB TLC AiFSSD composed of 16 AiFChips along with a host memory capacity of 8-GB. Each AiFChip achieves an effective bandwidth of 6.4 GB/s by employing two proposed flash read optimization techniques: cr-read and be-enc. The maximum bandwidth of AiFSSD for acceleration is 102.4 GB/s (6.4 GB/s × 16). To maximize generation throughput, AiF leverages head-level and tensor-level parallelism to enable concurrent execution with the host, as described in Section 5.1. The degree of tensor-level parallelism is limited by the available host memory capacity.

- **AiF--:** AiF-- operates identically to AiF, including parallel execution with the host, but does not support cr-read and be-enc techniques. In AiF--, each flash chip provides a bandwidth of 1.6 GB/s, limiting the maximum bandwidth of AiFSSD to 25.6 GB/s (1.6 GB/s × 16). Although ECC_LITE cannot perform error correction without be-enc, we assume error correction is feasible to focus on evaluating the performance of AiF--.

Table 2 summarizes the SSD configurations used in our evaluation. We assume a 1-TB SSD with eight flash channels and two chips per channel. For conventional SSDs with (2,3,2) coding, the

Table 2: Evaluated SSD configurations.

Configuration	8 channels; 2 chips/channel; 4 planes/chip; 16-KiB page
tR (2,3,2 coding)	LSB = 37 μ s; CSB = 46 μ s; MSB = 37 μ s;
tR (1,3,3 coding)	LSB = 28 μ s; CSB = 46 μ s; MSB = 46 μ s;
tR (cr-read)	9.7 μ s;
Bandwidth (PCIe)	8.0 GB/s external I/O bandwidth (PCIe 4.0, 4-lane);
Bandwidth (ONFI)	2.0 GB/s flash channel I/O bandwidth;

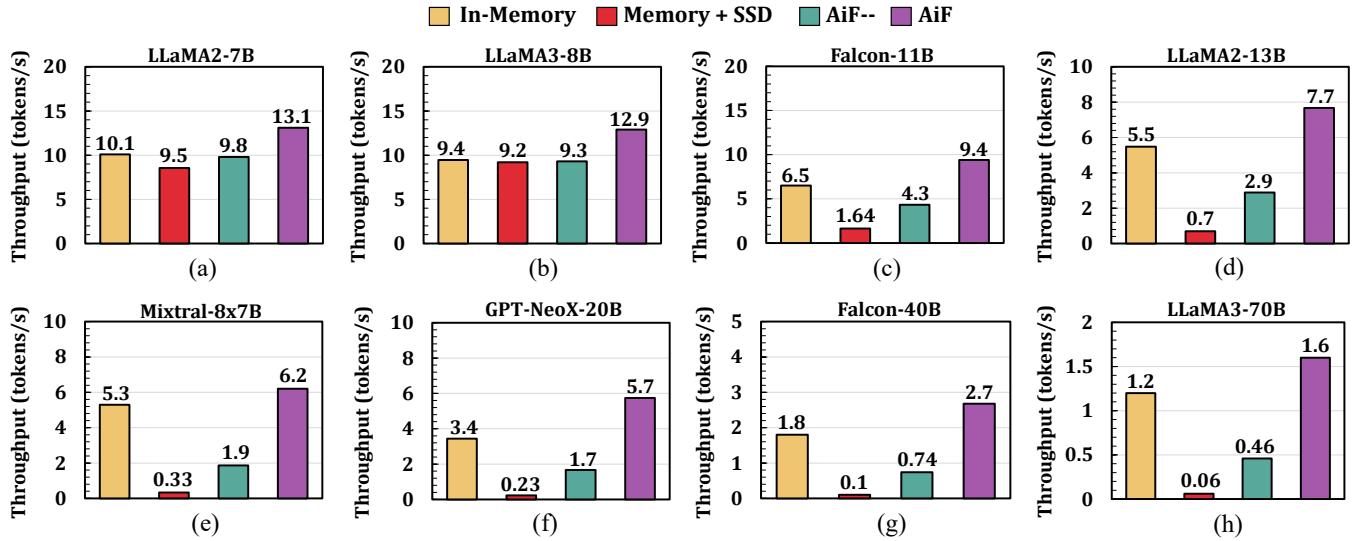


Figure 16: Throughput comparison of In-Memory, Memory+SSD, AiF--, and AiF.

Table 3: Evaluated LLM configurations.

Model	Weight (GiB)	KV Cache (GiB)	# Layers	# Heads
LLaMA2-7B [55]	6.7	1	32	32
LLaMA3-8B [57]	8	0.5	32	32
Falcon-11B [28]	11	0.25	40	32
LLaMA2-13B [54]	12.9	1.1	40	40
Mixtral-8x7B [2]	14.7 / 46.2	2	32	32
GPT-NeoX-20B [13]	20.4	1.05	44	64
Falcon-40B [27]	41.4	0.1	60	128
LLaMA3-70B [56]	69.8	1.3	80	80

tR values for LSB, CSB, and MSB pages are 37 µs, 46 µs, and 37 µs, respectively. For IFP blocks with (1,3,3) coding (i.e., with be-enc applied), the tR values are 28 µs, 46 µs, and 46 µs, respectively. The tR values are obtained from measurements on a real TLC flash chip. When cr-read is applied, the tR value for LSB pages within IFP blocks is reduced to 9.7 µs, based on evaluation results in Section 4.2.2. The external PCIe bandwidth is set to 8.0 GB/s, while each flash channel provides a bandwidth of 2.0 GB/s.

Evaluated Models. We evaluate eight distinct LLMs with various parameter sizes and architectures. Table 3 summarizes their key parameters and architectural details. All models are in INT8 format, the most commonly used format for on-device LLMs. Among these, Mixtral-8x7B adopts a mixture-of-experts (MoE) architecture, a variant of the transformer architecture. Unlike conventional transformers, which access the entire parameter set for each token generation, Mixtral-8x7B dynamically selects a subset of parameters at each step. As a result, while Mixtral-8x7B has a total weight size of 46.2-GiB, it requires reading only 14.7-GiB of parameters per token generation.

6.2 Evaluation Results

Throughput. Figure 16 presents the throughput comparison results across 8 different LLMs, highlighting four key observations. First, AiF effectively mitigates the performance bottleneck of the

baseline SSD offloading scheme. The Memory+SSD scheme operates efficiently for smaller models where parameters fit in memory (Figure 16(a) and (b)). However, as the model size grows (e.g., Figure 16(c)), a larger proportion of parameters must be offloaded to the SSD, leading to a sharp decline in throughput. In contrast, AiF sustains high throughput for larger models by leveraging the SSD’s high internal bandwidth. On average, AiF achieves 14.6x higher throughput than Memory+SSD across the evaluated models.

Second, AiF also outperforms in-memory inference. Despite being limited to 8-GB of memory, AiF achieves an average of 1.4x higher throughput than the In-Memory scheme. This is due to the AiFSSD’s high internal bandwidth of 102.4 GB/s, enhanced by cr-read and be-enc, which exceeds the memory bandwidth of 86.5 GB/s. Furthermore, AiF enables concurrent inference between the host and the AiFSSD, allowing the host memory bandwidth and the internal bandwidth of the AiFSSD to be utilized simultaneously. This advantage becomes more pronounced for smaller models, as a larger portion of the submatrix can fit in host memory, increasing the opportunities for concurrent inference.

Third, in-flash LLM inference without bandwidth optimization has limited scalability. The AiF-- scheme achieves 4.59x higher throughput than Memory+SSD on average by utilizing the internal bandwidth. However, its performance significantly declines for larger models, achieving only 0.74 tokens per second for a Falcon-40B model. In contrast, AiF sustains 2.7 tokens/s for the same model. AiF offers an average of 2.67x higher throughput than AiF--.

Fourth, MoE architecture aligns well with AiF. To fully leverage the benefits of MoE, both high storage capacity for large model parameters and sufficient bandwidth to access them quickly are essential. In-memory systems provide high bandwidth but are limited by their capacity, while SSDs offer large storage capacity but are constrained by low bandwidth. In contrast, AiFSSDs are particularly well-suited for MoE, as they combine high storage capacity with substantial internal bandwidth. Although AiF performance gradually declines as model size increases (e.g., Figures 16(h)), MoE

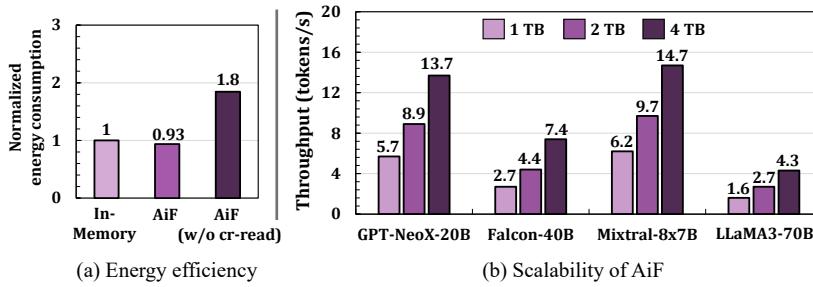


Figure 17: (a) Comparisons of energy efficiency. (b) Throughput scalability in AiF.

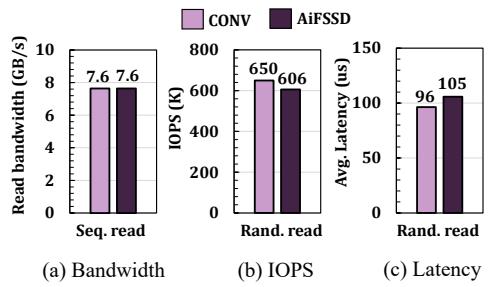


Figure 18: Overhead analysis.

enhances AiF’s scalability, enabling support for larger models while maintaining the same SSD capacity (Figure 16(e)).

Energy Efficiency. Next, we evaluate the energy efficiency of AiF and In-Memory. To understand the impact of cr-read, we also evaluate the energy consumption of AiF without cr-read applied. For the energy model, In-Memory assumes computations are performed on an energy-efficient NPU (1.4 TOPS/W as reported in [47]), and the host’s memory is modeled as LPDDR5 (7 pJ/bit based on [29]). Flash reads are modeled using the energy evaluation results from Figure 10, while ECC and GEMV operations are based on the results in Table 1. Flash channel and PCIe energy consumption are assumed to be 5.8 pJ/bit [66] and 7.5 pJ/bit [50], respectively. Figure 17(a) shows the energy consumption for generating 512 output tokens using the Falcon-40B model, normalized to the energy consumption of In-Memory. The results show that AiF achieves nearly a 2x improvement in energy efficiency when cr-read is applied. This improvement is attributed to parameter reads being the primary energy bottleneck in LLM inference, with cr-read reducing flash read energy consumption by more than 3x. Additionally, AiF and In-Memory exhibit comparable energy consumption, with AiF consuming approximately 7% less energy. This indicates that AiF is a more energy-efficient solution, particularly when considering the high static energy consumption of DRAM as its capacity increases.

Scalability. Figure 17(b) presents the throughput evaluation results for AiFSSD capacities of 1-TB, 2-TB, and 4-TB. While doubling the capacity theoretically doubles the internal bandwidth, the observed performance improvement scales by 1.35x to 1.68x. This sublinear scalability arises from two key factors: First, as shown in Figure 7, LLM inference involves multiple vector arithmetic operations interleaved with GEMV blocks. Consequently, reducing GEMV time alone does not result in a proportional reduction in overall inference time. Second, GEMV requests are handled through the NVMe protocol, which introduces unavoidable control overheads (e.g., interrupts, DMA) that hinder full utilization of internal bandwidth. Furthermore, as the number of flash chips increases with capacity, the control overhead grows, mainly due to the longer time required to load input vectors into flash chips via flash channels. To enhance AiF’s scalability, various scheduling optimizations on both the host and SSD sides are necessary. These further enhancements are left for future work.

Overhead Analysis. To analyze the overhead of be-enc, we compare the read performance of a conventional SSD (CONV) with (2,3,2) coding and the AiFSSD with (1,3,3) coding. For a worst-case analysis, we assume all blocks on the AiFSSD are IFP blocks. We

evaluate three key metrics for read performance: sequential read bandwidth, random read IOPS, and latency. Figure 18 presents the evaluation results, leading to two key observations. First, sequential read bandwidth remains stable despite the increased tR. As shown in Figure 18(a), both CONV and AiFSSD maintain a consistent bandwidth of 7.6 GB/s, due to the small external bandwidth relative to the large internal bandwidth of the SSD. Second, random reads exhibit slight performance degradation. In random read workloads, as shown in Figures 18(b) and (c), AiFSSD experiences a 6.8% reduction in IOPS and a 9.3% increase in latency compared to CONV. For random reads, only a fraction of the read page (4 KB out of 16 KB) is delivered to the host, alleviating the external bandwidth bottleneck and exposing the impact of the increased tR. These suggest that if AiFSSD contains a large number of IFP blocks, it may result in some performance loss in random read-intensive applications. We consider this a reasonable trade-off for the significant performance and reliability benefits be-enc provides for IFP.

7 Related Work

To our knowledge, this paper is the first to propose an IFP solution that improves the read bandwidth and reliability of flash chips to enable efficient in-flash LLM inference. We provide a brief overview of closely related work in two areas: 1) on-device LLM inference and 2) in-storage (in-flash) processing to improve the performance of data-intensive applications.

On-Device LLM Inference. Most research on on-device LLM inference has focused on model lightweighting techniques through quantization [9, 20, 51, 53, 81, 82, 87, 88]. However, maintaining accuracy while significantly reducing model size remains a considerable challenge and often demands extensive training efforts. Moreover, even with aggressive INT4 quantization, models exceeding 16B parameters still require more than 8-GB of memory. Some studies have sought to minimize the performance loss associated with offloading model parameters to SSDs by employing techniques such as I/O-computation pipelining [21], model collaboration [82], and sparsity optimization [3, 87]. While these approaches have shown performance improvements with limited memory capacity, they remain constrained by the bandwidth limitations of SSDs, making it challenging to perform inference on models larger than 10B parameters.

In-Storage Processing. To overcome the limited external bandwidth of SSDs, ISP has been widely explored across diverse application domains such as pattern matching [31, 33], query processing [10, 43, 59], data analytics [75, 91], genome processing [61],

graph processing [34, 48, 62], and DNN [22, 40, 79]. ISP solutions offer the advantage of reducing data transfer over the external PCIe bandwidth and leveraging resources within the SSD controller. However, the bandwidth limitations of the flash channels constrain the maximum achievable bandwidth gains of ISP, making it insufficient to handle high-bandwidth workloads such as LLM inference. **In-Flash Processing.** To overcome the bottleneck caused by flash channels and fully utilize the internal bandwidth of SSDs, some studies have explored IFP solutions, such as graph neural network [78], vector search [24, 77], DNN [38, 45], and bitwise processing [16]. While these efforts have successfully demonstrated the potential of IFP solutions, prior research has largely focused on integrating IFP into applications without adequately addressing practical challenges, such as ensuring reliability. The IFP-specialized techniques proposed by AiF for improving bandwidth and reliability can complement existing approaches, offering an orthogonal enhancement to previous work.

8 Conclusion

We have presented AiF, a novel IFP solution that integrates GEMV operations directly into flash chips, enabling the full utilization of SSDs' internal bandwidth for on-device LLM inference. The key feature of AiF is its IFP-specialized flash read techniques, specifically designed to meet the exceptionally high bandwidth and reliability demands of on-device LLM inference. By aligning the flash read procedure with the unique requirements of on-device LLM inference, AiF achieves the high generation throughput needed while maintaining inference accuracy through robust on-chip error correction with minimal implementation overhead. Our experimental results using a full-system AiF simulator demonstrate that AiF provides a 14.6x throughput improvement over the baseline SSD offloading scheme. Furthermore, AiF outperforms in-memory inference, achieving 1.4x higher throughput while significantly reducing memory footprint, demonstrating it as a practical and scalable solution for efficient on-device LLM deployment.

Acknowledgments

This work was supported in part by the Institute for Information & Communications Technology Planning & Evaluation (IITP) under Grant RS-2024-00347394, and in part by the Ministry of Science and ICT (MSIT), Republic of Korea, under Grant RS-2024-00456287 (Global Scholars Invitation Program). Myungsuk Kim was supported by the National Research Foundation of Korea (RS-2024-00414964). The ICT at Seoul National University provided research facilities for this study. (Corresponding author: Jihong Kim.)

References

- [1] 2020. *BSIM-CMG Technical Manual*. <http://bsim.berkeley.edu/models/bsimcmg/>. Accessed: Apr. 1, 2020. [Online]. Available: <http://bsim.berkeley.edu/models/bsimcmg/>.
- [2] Mistral AI. 2023. Mixtral-8x7B. <https://huggingface.co/mistralai/Mixtral-8x7B-v0.1>.
- [3] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifar, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. arXiv:2312.11514 [cs.CL].
- [4] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, Alessandro Cappelli, Ruxandra Cojocaru, Mérouane Debbah, Étienne Goffinet, Daniel Hesslow, Julien Launay, Quentin Malaric, Daniele Mazzotta, Badreddine Noune, Baptiste Pannier, and Guilherme Penedo. 2023. The Falcon Series of Open Language Models. arXiv:2311.16867 [cs.CL]. <https://arxiv.org/abs/2311.16867>
- [5] Marc Brysbaert. 2019. How many words do we read per minute? A review and meta-analysis of reading rate. *Journal of Memory and Language* 109 (2019), 104047.
- [6] Cadence. 2024. Spectre. https://www.cadence.com/en_US/home/tools/customic-analog-rf-design/circuit-design/virtuoso-schematic-editor.html.
- [7] Cadence. 2024. Virtuoso. https://www.cadence.com/en_US/home/tools/customic-analog-rf-design/circuit-design/virtuoso-schematic-editor.html.
- [8] Sungju Cho, Beomjun Kim, Hyunuk Cho, Gyeongseob Seo, Onur Mutlu, Myungsuk Kim, and Jisung Park. 2024. AERO: Adaptive Erase Operation for Improving Lifetime and Performance of Modern NAND Flash-Based SSDs. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3 (ASPLOS '24)*. Association for Computing Machinery, 101–118.
- [9] Tim Dettmers, Artidor Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. arXiv:2305.14314 [cs.LG]. <https://arxiv.org/abs/2305.14314>
- [10] Jaeyoung Do, Yang-Suk Kee, Jignesh M. Patel, Chanik Park, Kwanghyun Park, and David J. DeWitt. 2013. Query processing on smart SSDs: opportunities and challenges. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 1221–1230.
- [11] Juan P. Duarte, Sourabh Khandelwal, Aditya Medury, Chennming Hu, Pragya Kushwaha, Harshit Agarwal, Avirup Dasgupta, and Yogesh S. Chauhan. 2015. BSIM-CMG: Standard FinFET compact model for advanced circuit design. In *ESSCIRC Conference 2015 - 41st European Solid-State Circuits Conference (ESSCIRC)*, 196–201.
- [12] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. arXiv preprint arXiv:2407.21783 (2024).
- [13] EleutherAI. 2023. GPT-NeoX-20B. <https://huggingface.co/EleutherAI/gpt-neox-20b>.
- [14] NVMe Express. 2022. NVMe Namespace. <https://nvmexpress.org/resource/nvme-namespaces/>
- [15] Elias Frantar, Saleh Ashkboos, Torsten Hoefer, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs.LG]. <https://arxiv.org/abs/2210.17323>
- [16] Congming Gao, Xin Xin, Youyou Lu, Youtao Zhang, Jun Yang, and Jiwu Shu. 2021. ParaBit: Processing Parallel Bitwise Operations in NAND Flash Memory based SSDs. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 59–70.
- [17] Georg Gerganov. 2023. ggerganov/llama.cpp: Port of facebook's llama model in c/c++.
- [18] Donghyun Gouk, Miryeong Kwon, Hanyeoreum Bae, and Myoungsoo Jung. 2024. DockerSSD: Containerized In-Storage Processing and Hardware Acceleration for Computational SSDs. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 379–394.
- [19] Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. 2009. Characterizing flash memory: Anomalies, observations, and applications. In *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 24–33.
- [20] Cong Guo, Jiaming Tang, Weiming Hu, Jingwen Leng, Chen Zhang, Fan Yang, Yunxin Liu, Minyu Guo, and Yuahoo Zhu. 2023. OlivE: Accelerating Large Language Models via Hardware-friendly Outlier-Victim Pair Quantization. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23)*. ACM, 1–15.
- [21] Liwei Guo, Wonkyo Choe, and Felix Xiaozhu Lin. 2023. STI: Turbocharge NLP Inference at the Edge via Elastic Pipelining. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 791–803.
- [22] Saransh Gupta, Justin Morris, Mohsen Imani, Ranganathan Ramkumar, Jeffrey Yu, Aniket Tiwari, Baris Aksanli, and Tajana Simunic Rosing. 2020. THRIFTY: training with hyperdimensional computing across flash hierarchy. In *Proceedings of the 39th International Conference on Computer-Aided Design*.
- [23] Perttu Hämäläinen, Mikko Tavast, and Anton Kunnari. 2023. Evaluating Large Language Models in Generating Synthetic HCI Research Data: A Case Study.
- [24] Han-Wen Hu, Wei-Chen Wang, Yuan-Hao Chang, Yung-Chun Lee, Bo-Rong Lin, Huai-Mu Wang, Yen-Po Lin, Yu-Ming Huang, Chong-Ying Lee, Tzu-Hsiang Su, Chih-Chang Hsieh, Chia-Ming Hu, Yi-Ting Lai, Chung-Kuang Chen, Han-Sung Chen, Hsiang-Pang Li, Tei-Wei Kuo, Meng-Fan Chang, Keh-Chung Wang, Chun-Hsiung Hung, and Chih-Yuan Lu. 2022. ICE: An Intelligent Cognition Engine with 3D NAND-based In-Memory Computing for Vector Similarity Search Acceleration. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 763–783.
- [25] SK Hynix. 2022. SK Hynix Platinum P41. <https://www.techpowerup.com/ssd-specs/sk-hynix-platinum-p41-1-tb.d588>.

- [26] Jae-Woo Im, Woo-Pyo Jeong, Doo-Hyun Kim, Sang-Wan Nam, Dong-Kyo Shim, Myung-Hoon Choi, Hyun-Jun Yoon, Dae-Han Kim, You-Se Kim, Hyun-Wook Park, Dong-Hun Kwak, Sang-Won Park, Seok-Min Yoon, Wook-Ghee Hahn, Jin-Ho Ryu, Sang-Won Shim, Kyung-Tae Kang, Sung-Ho Choi, Jeong-Don Ihm, Young-Sun Min, In-Mo Kim, Doo-Sub Lee, Ji-Ho Cho, Oh-Suk Kwon, Ji-Sang Lee, Moo-Sung Kim, Sang-Hyun Joo, Jae-Hoon Jang, Sang-Won Hwang, Dae-Seok Byeon, Hyang-Ja Yang, Ki-Tae Park, Kye-Hyun Kyung, and Jeong-Hyuk Choi. 2015. 7.2 A 128Gb 3b/cell V-NAND flash memory with 1Gb/s I/O rate. In *2015 IEEE International Solid-State Circuits Conference - (ISSCC) Digest of Technical Papers*. 1–3.
- [27] Technology Innovation Institute. 2023. Falcon-40B. <https://huggingface.co/tiuae/falcon-40b>.
- [28] Technology Innovation Institute. 2024. Falcon-11B. <https://huggingface.co/tiuae/falcon-11b>.
- [29] Thomas Pawlowski J. 2019. Prospects for Memory. In *Workshop on Memory Centric High Performance Computing (MCHPC)*.
- [30] Hyun Sik Jeong and Seong Hwan Cho. 2021. Word-line and Charge-pump modeling of NAND Flash using Standard CMOS Logic Process. *Journal of Integrated Circuits and Systems* 7, 4 (2021).
- [31] Won Seob Jeong, Changmin Lee, Keunsoo Kim, Myung Kuk Yoon, Won Jeon, Myoungsoo Jung, and Won Woo Ro. 2020. REACT: Scalable and High-Performance Regular Expression Pattern Matching Accelerator for In-Storage Processing. *IEEE Transactions on Parallel and Distributed Systems* 31, 5 (2020), 1137–1151.
- [32] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. 2023. S³: Increasing GPU Utilization during Generative Inference for Higher Throughput. arXiv:2306.06000 [cs.AR] <https://arxiv.org/abs/2306.06000>
- [33] Sang-Woo Jun, Huy T. Nguyen, Vijay Gadepally, and Arvind. 2016. In-storage embedded accelerator for sparse pattern processing. In *2016 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–7.
- [34] Sang-Woo Jun, Andy Wright, Sizhuo Zhang, Shuotao Xu, and Arvind. 2018. GraBoost: Using Accelerated Flash Storage for External Graph Analytics. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. 411–424.
- [35] Ming-Yen Kao, Fredo Chavez, Sourabh Khandelwal, and Chenming Hu. 2022. Deep Learning-Based BSIM-CMG Parameter Extraction for 10-nm FinFET. *IEEE Transactions on Electron Devices* 69, 8 (2022), 4765–4768.
- [36] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. arXiv:2001.08361 [cs.LG] <https://arxiv.org/abs/2001.08361>
- [37] O. Khouri, R. Micheloni, A. Sacco, G. Campardo, and G. Torelli. 2000. Program word-line voltage generator for multilevel flash memories. In *ICECS 2000. 7th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.00EX445)*, Vol. 2. 1030–1033 vol.2.
- [38] Junkyum Kim, Myeonggu Kang, Yunki Han, Yang-Gon Kim, and Lee-Sup Kim. 2023. OptimStore: In-Storage Optimization of Large Scale DNNs with On-Die Processing. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 611–623.
- [39] Myungsuk Kim, Youngsun Song, Myoungsoo Jung, and Jihong Kim. 2018. SARO: A State-Aware Reliability Optimization Technique for High Density NAND Flash Memory. In *Proceedings of the 2018 Great Lakes Symposium on VLSI (GLSVLSI '18)*. 255–260.
- [40] Shine Kim, Yunho Jin, Gina Sohn, Jonghyun Bae, Tae Jun Ham, and Jae W. Lee. 2021. Behemoth: A Flash-centric Training Accelerator for Extreme-scale DNNs. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 371–385.
- [41] Sang-Hoon Kim, Jaehoon Shim, Eulidong Lee, Seongyeop Jeong, Ilkueon Kang, and Jin-Soo Kim. 2023. NVMeVirt: A Versatile Software-defined Virtual NVMe Device. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. Santa Clara, CA, 379–394.
- [42] Junnosuke Kondo and Toru Tanzawa. 2022. Pre-Emphasis Pulse Design for Reducing Bit-Line Access Time in NAND Flash Memory. *Electronics* 11 (2022).
- [43] Gunjae Koo, Kiran Kumar Matam, Te Li, H.V. Krishna Giri Narra, Jing Li, Hung-Wei Tseng, Steven Swanson, and Murali Annavaram. 2017. Summarizer: Trading Communication with Computing Near Storage. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 219–231.
- [44] Shiuhan-Hao Kuo. 2019. Ultra MMI : an LDPC decoder that doubles throughput at end-of-life.. In *Flash Memory Summit*.
- [45] Hunjun Lee, Minseop Kim, Dongmoon Min, Joonsung Kim, Jongwon Back, Honam Yoo, Jong-Ho Lee, and Jangwoo Kim. 2022. 3D-FPIM: An Extreme Energy-Efficient DNN Acceleration System Using 3D NAND Flash-Based In-Situ PIM Unit. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1359–1376.
- [46] Jaeyong Lee, Myungsunk Kim, Wonil Choi, Sanggu Lee, and Jihong Kim. 2022. TailCut: improving performance and lifetime of SSDs using pattern-aware state encoding. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) (DAC '22). 409–414.
- [47] Sang Min Lee, Hanjoon Kim, Jeseung Yeon, Juyun Lee, Younggeun Choi, Minho Kim, Changjae Park, Kiseok Jang, Youngsik Kim, Yongseung Kim, Changman Lee, Hyuck Han, Won Eung Kim, Rui Tang, and Joon Ho Baek. 2022. A 64-TOPS Energy-Efficient Tensor Accelerator in 14nm With Reconfigurable Fetch Network and Processing Fusion for Maximal Data Reuse. *IEEE Open Journal of the Solid-State Circuits Society* 2 (2022), 219–230.
- [48] Yunjae Lee, Jinha Chung, and Minsoo Rhu. 2022. SmartSAGE: training large-scale graph neural networks using in-storage processing architectures. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 932–945.
- [49] Qiao Li, Min Ye, Yufei Cui, Liang Shi, Xiaoqiang Li, Tei-Wei Kuo, and Chun Jason Xue. 2020. Shaving Retries with Sentinels for Fast Read over High-Density 3D Flash. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 483–495.
- [50] Shenggao Li, Fulvio Spagna, Ji Chen, Xiaoqing Wang, Luke Tong, Sujatha Gowder, Wenyan Jia, Roan Nicholson, Sita Iyer, Rui Song, Lily Li, Meng-hung Chen, Amanda Tran, Michael De Vita, Deepar Govindraj, Marcus Pasquarella, Dave Bradley, Frank Verdico, Matt Duwe, Eric Lee, and Michelle Wigton. 2018. A Power and Area Efficient 2.5-16 Gbps Gen4 PCIe PHY in 10nm FinFET CMOS. In *2018 IEEE Asian Solid-State Circuits Conference (A-SSCC)*. 5–8.
- [51] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. arXiv:2306.00978 [cs.CL] <https://arxiv.org/abs/2306.00978>
- [52] Li-Wei Liu, Mu-Hua Yuan, Yen-Chin Liao, and Hsie-Chia Chang. 2022. A 38.64-Gb/s Large-CPM 2-KB LDPC Decoder Implementation for nand Flash Memories. *IEEE Open Journal of Circuits and Systems* 3 (2022), 180–191.
- [53] Shih-yang Liu, Zechun Liu, Xijie Huang, Pingcheng Dong, and Kwang-Ting Cheng. 2023. LLM-FP4: 4-Bit Floating-Point Quantized Transformers. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 592–605.
- [54] Meta Llama. 2023. LLaMA2-13B. <https://huggingface.co/meta-llama/Llama-2-13b-hf>.
- [55] Meta Llama. 2023. LLaMA2-7B. <https://huggingface.co/meta-llama/Llama-2-7b-hf>.
- [56] Meta Llama. 2024. LLaMA3-70B. <https://huggingface.co/meta-llama/Meta-Llama-3-70B>.
- [57] Meta Llama. 2024. LLaMA3-8B. <https://huggingface.co/meta-llama/Meta-Llama-3-8B>.
- [58] Yina Lv, Liang Shi, Qiao Li, Congming Gao, Yunpeng Song, Longfei Luo, and Youtao Zhang. 2023. MGC: Multiple-Gray-Code for 3D NAND Flash based High-Density SSDs. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 122–136.
- [59] Vikram Sharma Mailthody, Zaid Qureshi, Weixin Liang, Ziyan Feng, Simon Garcia de Gonzalo, Youjie Li, Hubertus Franke, Jinjun Xiong, Jian Huang, and Wen-mei Hwu. 2019. DeepStore: In-Storage Acceleration for Intelligent Queries. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. 224–238.
- [60] Hikaru Makino and Toru Tanzawa. 2022. A 30% Power Reduction Circuit Design for NAND Flash by Utilizing 1.2V I/O Power Supply to Bitline Path. In *IEEE Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*. 10–13.
- [61] Nika Mansouri Ghiasi, Jisung Park, Harun Mustafa, Jeremie Kim, Ataberk Olgun, Arvid Gollwitzer, Damla Senol Cali, Can Firtina, Haiyu Mao, Nour Almadoun Alser, Rachata Ausavarungnirun, Nandita Vijaykumar, Mohammed Alser, and Onur Mutlu. 2022. 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*. 635–654.
- [62] Kiran Kumar Matam, Gunjae Koo, Haipeng Zha, Hung-Wei Tseng, and Murali Annavaram. 2019. GraphSSD: graph semantics aware SSD. In *Proceedings of the 46th International Symposium on Computer Architecture*. 116–128.
- [63] Rino Micheloni, Luca Crippa, and Alessia Marelli. 2010. *Inside NAND Flash Memories*. Springer.
- [64] Micron. 2023. Crucial T500. <https://www.techpowerup.com/ssd-specs/ crucial-t500-1-tb.d1771>.
- [65] Subrat Mishra, Hussam Amrouch, Jerin Joe, Chetan K. Dabhi, Karansingh Thakor, Yogesh S. Chauhan, Jörg Henkel, and Souvik Mahapatra. 2019. A Simulation Study of NBTI Impact on 14-nm Node FinFET Technology for Logic Applications: Device Degradation to Circuit-Level Interaction. *IEEE Transactions on Electron Devices* 66, 1 (2019), 271–278.
- [66] Daehoon Na, Jang-woo Lee, Seon-Kyoo Lee, Hwasuk Cho, Junha Lee, Manjae Yang, Eunjin Song, Anil Kavala, Tongsung Kim, Dong-Su Jang, Youngmin Jo, Ji-Yeon Shin, Byung-Kwan Chun, Tae-sung Lee, Byunghoon Jeong, Chi-Weon Yoon, Dongku Kang, Seungjae Lee, Jungdon Ihm, Dae Seok Byeon, Jinyub Lee, and Jai Hyuk Song. 2021. A 1.8-Gb/s/Pin 16-Tb NAND Flash Memory Multi-Chip Package With F-Chip for High-Performance and High-Capacity Storage. *IEEE Journal of Solid-State Circuits* 56, 4 (2021), 1129–1140.

- [67] Jisung Park, Jaeyong Jeong, Sungjin Lee, Youngsun Song, and Jihong Kim. 2016. Improving performance and lifetime of NAND storage systems using relaxed program sequence. In *Proceedings of the 53rd Annual Design Automation Conference (DAC '16)*. Association for Computing Machinery, Article 63, 6 pages.
- [68] Jisung Park, Myungsuk Kim, Myoungjun Chun, Lois Orosa, Jihong Kim, and Onur Mutlu. 2021. Reducing solid-state drive read latency by optimizing read-retry. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 702–716.
- [69] Jae-Woo Park, Doogon Kim, Sungwha Ok, Jaebeom Park, Taeheui Kwon, Hyunsoo Lee, Sungmook Lim, Sun-Young Jung, Hyeongjin Choi, Taikyu Kang, Gwan Park, Chul-Woo Yang, Jeong-Gil Choi, Gwihan Ko, Jaehyeon Shin, Ingoo Yang, Junghoon Nam, Hyeokchan Sohn, Seok-In Hong, Yohan Jeong, Sung-Wook Choi, Changwoon Choi, Hyun-Soo Shin, Junyoung Lim, Dongkyu Youn, Sanghyuk Nam, Juyeab Lee, Myungkyu Ahn, Hoseok Lee, Seungpil Lee, Jongmin Park, Kichang Gwon, Woopyo Jung, Jungdal Choi, Jinkook Kim, and Kyo-Won Jin. 2021. 30.1 A 176-Stacked 512Gb 3b/Cell 3D-NAND Flash with 10.8Gb/mm² Density with a Peripheral Circuit Under Cell Array Architecture. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, Vol. 64. 422–423.
- [70] Sung-Ho Park, Dongseok Kwon, Ho-Nam Yoo, Jong-Won Back, Joon Hwang, Yeongheon Yang, Jae-Joon Kim, and Jong-Ho Lee. 2022. Retention Improvement in Vertical NAND Flash Memory Using 1-bit Soft Erase Scheme and its Effects on Neural Networks. In *2022 International Electron Devices Meeting (IEDM)*. 5.5.1–5.5.4.
- [71] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative LLM inference using phase splitting. arXiv:2311.18677 [cs.AR] <https://arxiv.org/abs/2311.18677>
- [72] Samsung. 2022. Samsung 990 PRO. <https://www.techpowerup.com/ssd-specs/samsung-990-pro-1-tb.d861>.
- [73] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. arXiv:1909.08053 [cs.CL] <https://arxiv.org/abs/1909.08053>
- [74] Synopsys. 2024. Design Compiler. <https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html>.
- [75] Mahdi Torabzadehkashi, Siavash Rezaei, Ali Heydarigorji, Hosein Bobarshad, Vladimir Alves, and Nader Bagherzadeh. 2019. Catalina: In-Storage Processing Acceleration for Scalable Big Data Analytics. In *2019 27th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*. 430–437.
- [76] Chien-Ting Tung and Chenming Hu. 2023. Neural Network-Based BSIM Transistor Model Framework: Currents, Charges, Variability, and Circuit Simulation. *IEEE Transactions on Electron Devices* 70, 4 (2023), 2157–2160.
- [77] Yitu Wang, Shiyu Li, Qilin Zheng, Linghao Song, Zongwang Li, Andrew Chang, Hai "Helen" Li, and Yiran Chen. 2024. NDSEARCH: Accelerating Graph-Traversal-Based Approximate Nearest Neighbor Search through Near Data Processing. arXiv:2312.03141 [cs.AR]
- [78] Yuyue Wang, Xirui Pan, Yuda An, Jie Zhang, and Glenn Reinman. 2024. Beacon-GNN: Large-Scale GNN Acceleration with Out-of-Order Streaming In-Storage Computing. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 330–344.
- [79] Mark Wilkening, Udit Gupta, Samuel Hsia, Caroline Trippel, Carole-Jean Wu, David Brooks, and Gu-Yeon Wei. 2021. 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 (ASPLOS '21)*. 717–729.
- [80] Samkyu Won, Yujong Noh, Hyunchul Cho, Jeil Ryu, Sung-Un Choi, Sungdae Choi, DuckJu Kim, Junseop Chung, Bong-Seok Han, and Eui-Young Chung. 2011. High-voltage wordline generator for low-power program operation in NAND flash memories. *IEEE Asian Solid-State Circuits Conference 2011* (2011), 169–172.
- [81] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2024. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. arXiv:2211.10438 [cs.CL] <https://arxiv.org/abs/2211.10438>
- [82] Daliang Xu, Wangsong Yin, Xin Jin, Ying Zhang, Shiyun Wei, Mengwei Xu, and Xuanze Liu. 2023. LLMcad: Fast and Scalable On-device Large Language Model Inference. arXiv:2309.04255 [cs.NI]
- [83] Weihong Xu, Jaeyoung Kang, and Tajana Rosing. 2022. A near-storage framework for boosted data preprocessing of mass spectrum clustering. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 313–318.
- [84] Li Yan, Hsu Cynthia, and Oowada Ken. 2014. Non-Volatile Memory And Method With Improved First Pass Programming. U.S Patent 8811091, 2014.
- [85] Ziye Yang, James R. Harris, Benjamin Walker, Daniel Verkamp, Changpeng Liu, Cunyin Chang, Gang Cao, Jonathan Stern, Vishal Verma, and Luse E. Paul. 2017. SPDK: A Development Kit to Build High Performance Storage Applications. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 154–161.
- [86] Min Ye, Qiao Li, Congming Gao, Shun Deng, Tei-Wei Kuo, and Chun Jason Xue. 2022. Stop unnecessary refreshing: extending 3D NAND flash lifetime with ORBER. *CCF Trans. High Perform. Comput.* 4, 3 (2022), 281–301.
- [87] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. 2023. EdgeMoE: Fast On-Device Inference of MoE-based Large Language Models.
- [88] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. 2023. RPTQ: Reorder-based Post-training Quantization for Large Language Models. arXiv:2304.01089 [cs.CL] <https://arxiv.org/abs/2304.01089>
- [89] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. HellaSwag: Can a Machine Really Finish Your Sentence? <https://arxiv.org/abs/1905.07830>
- [90] Kai Zhao, Wenzhe Zhao, Hongbin Sun, Xiaodong Zhang, Nanning Zheng, and Tong Zhang. 2013. LDPC-in-SSD: Making Advanced Error Correction Codes Work Effectively in Solid State Drives. In *11th USENIX Conference on File and Storage Technologies (FAST 13)*. 243–256.
- [91] Chen Zou and Andrew A. Chien. 2022. ASSASIN: Architecture Support for Stream Computing to Accelerate Computational Storage. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 354–368.