

Exploring High-Bandwidth Flash for Modern LLM Inference: Opportunities and Challenges

Dowon Son, Yonggon Park, *Graduate Student Member, IEEE*, Hyunuk Cho^{ID}, Hyungkyu Ham^{ID},
Onur Mutlu^{ID}, *Fellow, IEEE*, Sungjin Lee, Gwangsun Kim^{ID}, *Member, IEEE*, and Jisung Park^{ID}, *Member, IEEE*

Abstract—This work investigates the potential benefits and technical challenges of using high-bandwidth flash (HBF) for large language model (LLM) inference. HBF has gained increasing attention as a promising solution to mitigate memory-capacity bottlenecks in modern LLM-serving systems, but its benefits and challenges remain largely uninvestigated. To address this gap, we thoroughly analyze HBF-based LLM-serving systems under diverse system configurations and operating scenarios in which HBF serves as a main GPU-memory component to handle both reads and writes. Our analysis shows that, despite its limited write performance, HBF can significantly improve the batch size, throughput, and flexibility of LLM-serving systems while reducing the minimum GPU requirements, but realizing these benefits critically depends on sustaining HBM-comparable read bandwidth and requires significant endurance improvements.

Index Terms—High-bandwidth flash, LLM-serving systems.

I. INTRODUCTION

MODERN computing systems for LLM inference suffer from the limited capacity of HBM. To improve inference quality, recent LLMs continue to grow rapidly in model size and context length, far outpacing HBM-capacity scaling. A common solution to this mismatch is to scale aggregate HBM capacity with multiple GPUs, which (i) significantly increases system cost by requiring many GPUs and expensive high-speed interconnects and (ii) limits system flexibility by tightly coupling multiple GPUs to serve each query.

HBF has gained increasing attention as a promising solution to mitigate the memory-capacity bottleneck. It replaces HBM's DRAM with low-latency NAND flash dies, significantly increasing per-stack capacity by more than $14\times$ (e.g., 512 GB) compared to HBM4 (36 GB). By exploiting massive die- and plane-level parallelism, HBF aims to deliver per-stack read bandwidth comparable to HBM4 (e.g., 1.6 TB/s [1]).

In this work, we present the first systematic study on the opportunities and challenges of using HBF to *replace* (most of) HBM in modern LLM-serving systems. Despite the significant capacity advantage over HBM, HBF's effectiveness as a main GPU-memory component in practical LLM-serving scenarios remains largely unexplored. To address this, we thoroughly analyze HBF-based LLM serving with recent large-scale models while varying GPU counts, target service-level objectives (SLOs), and average context lengths.

Based on our analysis, we make ten key observations and five takeaways on the performance, bottlenecks, and design implications of HBF-based LLM-serving systems. We highlight three key findings. First, HBF has strong potential to significantly improve the batch size,

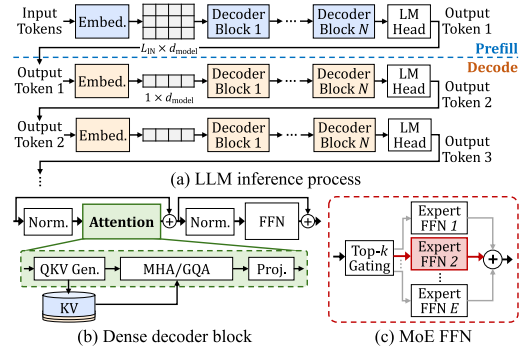


Fig. 1. LLM inference process and architecture.

throughput, and flexibility of modern LLM-serving systems, despite its limited write performance. Second, the larger memory capacity alone is insufficient to realize these benefits; sustaining HBM-comparable read bandwidth is also critical under SLO constraints. Third, in HBF-based systems, KV-cache writes introduce non-trivial performance overheads and significant endurance challenges.

II. BACKGROUND AND MOTIVATION

LLM Inference: Fig. 1(a) depicts the overall process of LLM inference, which consists of two phases: (i) *prefill* and (ii) *decode*. Both phases pass through the same token-embedding layer, decoder blocks, and language-model (LM) head. Given a query with L_{IN} tokens, the prefill phase processes all tokens in parallel to generate the first output token. The decode phase then generates L_{OUT} tokens *autoregressively*, using each output token as a new input token for the next iteration.

The decoder blocks iteratively refine each token's representation, i.e., a d_{model} -dimensional hidden vector, whereas the token-embedding layer and LM head map between tokens and vectors. Each decoder block contains two key components: (i) an *attention layer* and (ii) a *feedforward network* (FFN) layer (Fig. 1(b)). The attention layer projects input representations into query, key, and value (QKV) tensors and aggregates information via multi-head attention (MHA); it is common practice to cache all key and value tensors (KV cache) to avoid redundant recomputation in the decode phase. The FFN layer applies position-wise nonlinear transformations independently to each token. In dense models (e.g., Llama 3 405B [2]), a single large FFN processes all tokens, whereas recent mixture-of-experts (MoE) models (e.g., Llama 4 Maverick [3]) use multiple smaller expert FFNs and route each token to only a few selected ones, enabling large total model capacity while limiting per-token computation and memory access (Fig. 1(c)).

High-Bandwidth Flash: Since its introduction in early 2025 [4], HBF has attracted growing attention as a potential solution to the memory-capacity bottleneck in modern LLM-serving systems. Recent LLMs have rapidly grown far beyond a single GPU's memory capacity (e.g., Llama 4 Maverick's model size is 746 GB, whereas a high-end

Received 11 May 2026; revised 9 June 2026; accepted 15 June 2026. Date of publication 22 June 2026; date of current version 30 June 2026. This work was supported in part by the NRF under Grant RS-2025-00519994 and Grant RS-2024-00415602 and in part by IITP under Grant RS-2026-25529760, Grant RS-2024-00437866, and Grant RS-2024-00347394 of Korea. (Corresponding author: Jisung Park.)

Dowon Son, Yonggon Park, Hyunuk Cho, Hyungkyu Ham, Sungjin Lee, Gwangsun Kim, and Jisung Park are with POSTECH, Pohang-si 37673, South Korea (e-mail: jisung.park@postech.ac.kr).

Onur Mutlu is with ETH Zürich, 8092 Zurich, Switzerland.

Digital Object Identifier 10.1109/LCA.2026.3705817

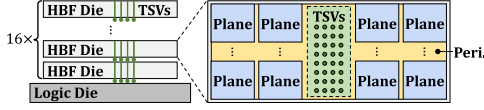


Fig. 2. Expected organization of HBF.

TABLE I
GPU-MEMORY CONFIGURATIONS OF EVALUATED SYSTEMS

GPU	# Stacks (HBM/HBF)	Capacity [GB]	RD BW [TB/s] (HBM/HBF)	WR BW [TB/s] (HBM/HBF)
HBM4	8/0	288	12.8/-	12.8/-
CONV	1/7	3,620	1.6/2.45	1.6/0.073
CONV ⁺	0/8	4,096	-/2.80	-/0.084
HBF	1/7	3,620	1.6/11.2	1.6/0.112
HBF ⁺	0/8	4,096	-/12.8	-/0.128

GPU [5] provides only 288 GB), making multi-GPU deployment inevitable. Although various optimizations such as quantization [6] and grouped-query attention (GQA) [7] significantly reduce KV-cache size, the maximum batch size remains limited due to continuously increasing model size and context length. HBF has the potential to fundamentally address this bottleneck by exploiting the high storage density of NAND flash memory.

Fig. 2 shows the high-level organization of an HBF stack, which vertically stacks multiple core dies on a logic base die via TSVs, as in conventional HBM. Each core die contains multiple low-latency NAND flash planes that operate concurrently, thereby providing high read bandwidth through massive parallelism. According to Sandisk’s projection, HBF targets μ s-order latency for 4-KiB reads while delivering per-stack bandwidth comparable to that of HBM4 (e.g., 1.6 TB/s) [1]. An HBF stack is expected to consume higher power (< 80 W) compared to an HBM stack (40 W) [1], which could also cause thermal issues. However, we believe that this is primarily a power-delivery and thermal-management challenge rather than a fundamental limitation of HBF. Note that, as we demonstrate in this work, HBF has the potential to reduce the GPU count required for serving large-scale LLMs, which may help offset its higher per-stack power consumption at the system level.

Our goal in this work is to provide new insights into HBF’s practical benefits and technical challenges in modern LLM-serving systems, which remain largely underexplored despite HBF’s high potential. To our knowledge, only one recent work has shown HBF’s effectiveness at improving the efficiency of cache-augmented generation (CAG) when used as a read-only memory extension to HBM [8], but it is unclear how HBF can benefit more general LLM-serving scenarios as a main GPU-memory component. To address this, we thoroughly analyze the effectiveness of HBF-enabled LLM-serving systems with various system configurations and workloads.

III. METHODOLOGY

Hardware Configuration: Table I summarizes the GPU-memory configurations of five LLM-serving systems we evaluate, which share the same state-of-the-art GPU-core architecture [5]. First, HBM4 contains eight HBM stacks per GPU, each providing 36-GB capacity and 1.6-TB/s read bandwidth.

Second, HBF and HBF⁺ replace seven and all eight HBM stacks of the GPUs in HBM4, respectively, with HBF stacks offering 512 GB of capacity and 1.6 TB/s of read bandwidth per stack as projected by Sandisk; to avoid excessive writes to NAND flash memory, we assume that HBF reserves one HBM stack to store intermediate data, whereas HBF⁺ leverages 40-MB SRAM on the logic die instead.¹ Our analysis shows that much of the intermediate data has a short lifetime, which enables quick release of SRAM-buffer space and thus allows HBF⁺ to

support a large batch size (e.g., 327 under Llama 4 Maverick) despite the limited SRAM capacity (320 MB per GPU). Both HBF and HBF⁺ place model weights and KV cache in NAND flash, as they require a large storage capacity and exhibit read-dominant access patterns. To achieve the target read bandwidth (1.6 TB/s per stack), we assume 25 planes per HBF core die and a 1- μ s 4-KiB page-read latency. As in prior work [8], we further assume that each HBF stack uses a dedicated 3.13-MB SRAM staging buffer on its logic die to prefetch and double-buffer read data, thereby minimizing the performance impact of μ s-scale read latency.

Third, CONV and CONV⁺ are identical to HBF and HBF⁺, respectively, except for each stack’s read and write bandwidth. We introduce these conservative configurations since achieving HBF’s target bandwidth requires significant advancements over existing low-latency NAND flash technologies, e.g., Samsung’s Z-NAND [9] and Kioxia’s XL-Flash [10] that provide only 87.4-GB/s and 262-GB/s read bandwidth with 16 dies, respectively. Although future HBF could achieve the target bandwidth through (i) finer-grained plane partitioning, which would reduce read latency and increase internal parallelism, and (ii) wafer-bonding-based integration [11], which would mitigate the area overhead of aggressive partitioning, we project CONV and CONV⁺ from existing low-latency NAND flash technologies. We assume 16 planes per die [10] and a 3- μ s page-read latency [9], achieving per-stack read bandwidth of 0.35 TB/s. Since HBF’s expected write performance is not publicly available, we assume a 100- μ s page-program latency for all HBF configurations based on [9].

Simulation Methodology: We use an existing simulator for LLM-serving systems, called LLMSimulator [12], which we extend in three ways. First, we add an HBF-aware timing model that individually configures read and write bandwidths to reflect the performance asymmetry of NAND flash memory. Second, we improve the modeling of continuous batching and disaggregated prefill-decode execution by accounting for the performance overhead of KV-cache writes when transferring outputs from prefill to decode nodes. Third, we implement key techniques widely adopted in modern LLMs and LLM-serving systems, including chunked attention and pipeline parallelism. We will open-source our extended LLM simulator.

We make four assumptions about the system configuration and execution model in our analysis. First, we focus on decode nodes in disaggregated prefill-decode execution, as the prefill phase is inherently compute-bound and thus benefits little from HBF’s large capacity. Second, we assume continuous batching that maintains a full batch by admitting new queries as in-flight ones complete. Third, each evaluated system selects the parallelism configuration that maximizes the achievable system throughput subject to all constraints, including SLO requirements (e.g., time per output token (TPOT) ≤ 0.1 s), GPU-memory capacity, and on-die SRAM limits (in HBF⁺ and CONV⁺), by combining data, tensor, pipeline, and expert parallelism. Fourth, all evaluated systems are equipped with high-performance interconnects: NVLink (1,800 GB/s) for intra-node GPU communication (up to eight GPUs per node) and InfiniBand (100 GB/s) for inter-node communication.

Workloads: We evaluate Llama 3 405B (dense) and Llama 4 Maverick (MoE) across three application scenarios with different context-length characteristics; we vary each query’s $\langle L_{IN}, L_{OUT} \rangle$ values across $\langle 1,660, 373 \rangle$, $\langle 5.9K, 499 \rangle$, and $\langle 103.5K, 1.1K \rangle$ based on the average values in the ShareGPT [13] (Short), LongBench [14] (Mid), and English summarization [15] (Long), respectively. We model a constant query arrival rate matching the completion rate, which enables steady-state performance analysis under continuous batching.

IV. EVALUATION RESULTS

Batch Size: Fig. 3 compares the maximum *per-GPU* batch sizes of five LLM-serving systems under 0.1-second TPOT SLO across GPU counts ranging from 1 to 16. We compactly encode results for different

¹We find that writing intermediate data increases write traffic by up to 100 $\times$ for every decode iteration, incurring significant endurance and performance overheads. We use the SRAM capacity from prior work [8].

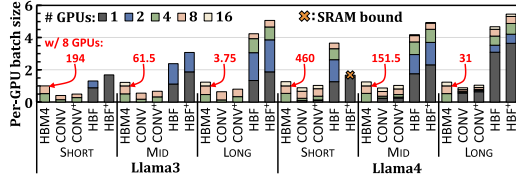


Fig. 3. Per-GPU batch size under 0.1-second TPOT SLO.

GPU counts within a single bar; the top of each colored segment indicates the maximum per-GPU batch size at the corresponding GPU count. If a larger GPU count does not increase per-GPU batch size, the corresponding segment does not appear in Fig. 3 because it is subsumed by the preceding segment. All values are normalized to 8-GPU HBM4 for each workload, whose values are denoted in Fig. 3.

We make four key observations from Fig. 3. First, HBF and HBF⁺ enable significantly larger batch sizes than HBM4 at the same GPU count while meeting the TPOT SLO across all workloads, e.g., providing 1.3–4.5 $\times$ and 1.7–5.3 $\times$ gains, respectively, in the 8-GPU configuration. In particular, HBM4 requires at least four GPUs to serve both LLMs (*no* 1-/2-GPU segments in *all* HBM4 bars), whereas 1-GPU HBF and HBF⁺ support a larger per-GPU batch size than 8-GPU HBM4 in most cases. The results suggest that HBF has high potential to enable more cost-effective LLM-serving systems by substantially reducing the minimum GPU requirement.

Second, CONV and CONV⁺ support only limited batch sizes, often even smaller than those of HBM4. Their relatively low read bandwidths make it challenging to meet the TPOT SLO with large batches, leading to significant underutilization of the HBF capacity (e.g., up to 95% with Llama3 under Short). The results demonstrate that increasing memory capacity alone is insufficient for practical HBF-based LLM-serving systems; sustaining read bandwidth comparable to HBM is critical to take advantage of the expanded capacity.

Third, dedicating an HBM stack to intermediate data limits the batch-size improvements of HBF-based systems, e.g., with eight GPUs, HBF⁺ can support 24% larger batch sizes on average than HBF in Mid. In HBF, the single HBM stack stores only a limited amount of data, so its bandwidth remains largely underutilized while the HBF stacks handle most data accesses. These results again highlight the importance of read bandwidth to fully exploit HBF under SLO constraints.

Fourth, HBF and HBF⁺ provide increasing benefits with the context length, e.g., HBF⁺ achieves 124%/200% higher average gains over HBM4 in Mid/Long than in Short. In general, LLM-serving systems can support larger batches for shorter-context queries, which limits HBF's benefits in three aspects; (i) the inter-GPU communication increases almost linearly with batch size, making it difficult to further scale the already large batch sizes in short-context workloads under the TPOT SLO; (ii) in the dense model, FFN execution shifts toward compute-bound GEMM (general matrix multiplication) operations, which also limits further batch-size scaling under the SLO; (iii) a larger batch size also linearly increases the peak size of intermediate data, so the limited SRAM-buffer capacity (e.g., 40 MB) bottlenecks batch-size scaling in HBF⁺.

Takeaway 1: HBF has the potential to significantly increase batch sizes in modern LLM inference, but its practical benefits over HBM depend critically on its read bandwidth.

System Throughput: Fig. 4 compares the per-GPU token-generation throughput, i.e., tokens per second (TPS), of five LLM-serving systems under 0.1-second TPOT SLO across GPU counts ranging from 1 to 16.

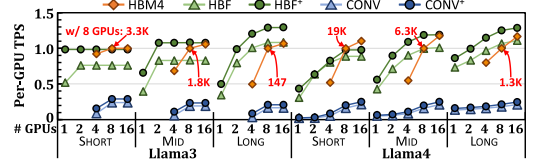


Fig. 4. System throughput of evaluated LLM-serving systems.

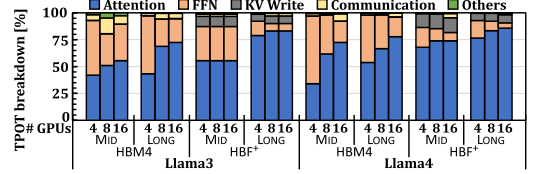


Fig. 5. Runtime breakdown under 0.1-second TPOT SLO.

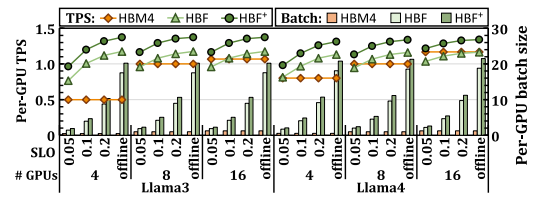


Fig. 6. Performance and batch size impact of TPOT SLO.

All values are normalized to the 8-GPU HBM4 (denoted) for each workload.

We make two key observations from Fig. 4. First, for each evaluated LLM and workload, both HBF and HBF⁺ provide substantially higher per-GPU TPS than HBM4 at one or more GPU counts, despite their limited write performance. Although their gains diminish as GPU count increases, this does *not* imply that HBF benefits only small-scale systems. In fact, HBM4 also eventually exhibits diminishing TPS gains from GPU scaling, as shown in Fig. 4; larger batches improve TPS by better amortizing model-weight accesses across more queries, but beyond a sufficient batch size, the benefits severely decrease as the bottleneck shifts to other operations whose costs increase with the batch size, e.g., KV-cache accesses and inter-GPU communication. HBF and HBF⁺ consistently achieve such batch sizes with fewer GPUs and thus provide significant TPS gains until HBM4 is scaled to sufficient GPUs, which suggests that HBF can benefit a wide range of LLM-serving scenarios, e.g., enabling large-scale systems to better cope with the continuous, rapid growth in model size.

Second, HBF⁺ can provide higher per-GPU TPS than HBM4 *even with fewer GPUs*. For example, in the Llama4 Long workload, the per-GPU TPS of 4-GPU HBF⁺ is 15% higher than that of 8-GPU HBM4, highlighting HBF's potential to improve the flexibility of LLM-serving systems; with eight GPUs, HBF⁺ can run two independent 4-GPU instances with higher aggregate throughput than 8-GPU HBM4, while reducing scheduling complexity and improving fault isolation.

Takeaway 2: Despite its limited write performance, HBF has the potential to improve the throughput and flexibility of LLM-serving systems while reducing the minimum GPU requirement, when used as a main GPU-memory component.

Performance Breakdown: To better understand the source of HBF's benefits over HBM, we analyze the latency breakdown of HBF⁺ and HBM4 under 0.1-second TPOT SLO. Fig. 5 shows the fraction of decoding time spent on each component. 'KV Write' represents writing

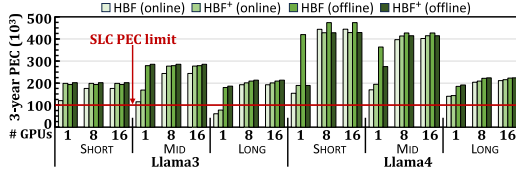


Fig. 7. Required PEC under different operating scenarios.

the KV cache for *newly admitted queries* under continuous batching,² and ‘Others’ includes layer normalization, residual, and LM head.

We make two key observations from Fig. 5. First, KV-cache writes have a non-trivial performance impact in HBF⁺, e.g., accounting for 5–13.9% of the execution time in Llama4. The impact increases with more GPUs, shorter contexts, and the MoE model, due to more frequent query admissions. The results suggest that HBF⁺ could provide even greater benefits over HBM4 with improved write performance.

Second, HBF⁺ significantly reduces the fraction of decoding time spent on FFN layers and inter-GPU communications. Compared to HBM4, HBF⁺ consistently shifts the performance bottleneck toward attention layers, suggesting that it (i) more effectively amortizes model-weight accesses across queries with larger batches and (ii) reduces communication overheads by leveraging less tightly coupled inter-GPU parallelism. In attention layers, massive KV-cache reads dominate execution time, diminishing the benefits of large batches.

Takeaway 3: Improving KV-cache write performance can further increase the benefits of HBF-based systems.

Effect of SLO: We evaluate how HBF’s effectiveness changes depending on the TPOT SLO. Fig. 6 compares the per-GPU TPS (corresponding to the left y -axis) and batch size (the right y -axis) of three LLM-serving systems, which we measure in the Long workload while varying TPOT SLOs. We also evaluate an offline inference scenario, where each query only needs to complete within 24 hours (i.e., no TPOT SLO). All values are normalized to the 8-GPU HBM4 for each workload.

We observe that the target SLO significantly affects HBF’s benefits; the more relaxed the SLO, the greater the benefits HBF provides. Although HBF⁺ always outperforms HBM4 even under 0.05-second TPOT SLO in Long, its TPS gains substantially increase in the offline scenario even at large GPU counts, e.g., from 4.1% to 14.8% in Llama4 with 16 GPUs. The results highlight the distinct performance bottlenecks in HBM- and HBF-based systems; HBM4’s per-GPU TPS and batch size hardly change across SLOs due to the memory-capacity bottleneck, whereas HBF and HBF⁺ significantly benefit from relaxed SLOs by leveraging their large capacity.

Takeaway 4: HBF is especially promising for throughput-oriented or offline LLM inference with relaxed target SLOs.

Write Traffic Analysis: We analyze the write traffic to HBF to understand the endurance requirement in HBF-based LLM-serving systems; NAND flash memory has limited endurance, e.g., a single-level cell (SLC) block cannot guarantee data reliability after experiencing 100 K program and erase (P/E) cycles. Fig. 7 shows the per-block

P/E-cycle count (PEC) of HBF and HBF⁺ over three years, a common GPU-warranty period, under various operating scenarios.

We observe that both HBF and HBF⁺ incur significant P/E-cycle counts far exceeding 100 K in most cases. In general, the write traffic increases more sharply when the systems can achieve higher TPS, i.e., with shorter context lengths, more GPUs, relaxed target SLO, and the MoE model, as every processed token generates its own KV-cache tensors. Writing the KV cache in a retention-relaxed manner [16] could mitigate the endurance problem, but we expect that significant improvements in NAND flash endurance are still critical to prevent the premature replacement of costly GPUs.

Takeaway 5: NAND flash endurance remains a critical challenge for practical HBF-based LLM-serving systems.

V. CONCLUSION

We present the first study on the opportunities and challenges of HBF as an alternative to HBM in modern LLM-serving systems. Our analysis shows that HBF has strong potential to improve the throughput and flexibility while reducing the minimum GPU requirements, but it also requires HBM-comparable read bandwidth as well as significant improvements in write performance and endurance. We hope that our findings help guide future research toward efficient and practical HBF-based LLM-serving systems.

REFERENCES

- [1] X. Ma et al., “Challenges and research directions for large language model inference hardware,” 2026, *arXiv:2601.05047*.
- [2] A. Grattafiori et al., “The Llama 3 Herd of models,” 2024, *arXiv:2407.21783*.
- [3] Meta, “The Llama 4 Herd: The beginning of a new era of natively multimodal AI innovation,” 2025. [Online]. Available: <https://ai.meta.com/blog/llama-4-multimodal-intelligence/>
- [4] Sandisk, “Future FWD: Sandisk 2025 investor day,” 2025. [Online]. Available: <https://investor.sandisk.com/static-files/79481580-ada2-4e08-bdeb-4b440d08f4ab>
- [5] NVIDIA, “NVIDIA DGX Rubin NVL8,” 2026. [Online]. Available: <https://www.nvidia.com/en-us/data-center/dgx-rubin-nvl8/>
- [6] C. Hooper et al., “KVQuant: Towards 10 million context length LLM Inference with KV cache quantization,” 2025, *arXiv:2401.18079*.
- [7] J. Ainslie et al., “GQA: Training generalized multi-query transformer models from multi-head checkpoints,” in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2023, pp. 4895–4901.
- [8] M. Ha et al., “H3: Hybrid architecture using high bandwidth memory and high bandwidth flash for cost-efficient LLM inference,” *IEEE Comput. Arch. Lett.*, vol. 25, no. 1, pp. 49–52, Jan./Jun. 2026.
- [9] W. Cheong et al., “A flash memory controller for 15 μ s ultra-low-latency SSD using high-speed 3D NAND flash with 3 μ s read time,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2018, pp. 338–340.
- [10] T. Kouchi et al., “A 128Gb 1b/Cell 96-word-Line-layer 3D flash memory to improve random read latency with tPROG=75 μ s and tR=4 μ s,” in *Proc. IEEE Int. Solid-State Circuits Conf.*, 2020, pp. 226–228.
- [11] Sandisk, “Sandisk-HBF-Fact-Sheet,” 2025. [Online]. Available: https://documents.sandisk.com/content/dam/asset-library/en_us/assets/public/sandisk/collateral/company/Sandisk-HBF-Fact-Sheet.pdf
- [12] S. Yun et al., “Duplex: A device for large language models with mixture of experts, grouped query attention, and continuous batching,” in *Proc. Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2024, pp. 1429–1443.
- [13] “ShareGPT: GPT4,” 2024. [Online]. Available: https://huggingface.co/datasets/shibing624/sharegpt_gpt4
- [14] Y. Bai et al., “LongBench: A bilingual, multitask benchmark for long context understanding,” 2024, *arXiv:2308.14508*.
- [15] X. Zhang et al., “ ∞ Bench: Extending long context evaluation beyond 100K tokens,” 2024, *arXiv:2402.13718*.
- [16] X. Pan et al., “InstAttention: In-storage attention offloading for cost-effective long-context LLM inference,” in *Proc. IEEE Int. Symp. High Perform. Comput. Architecture*, 2025, pp. 1510–1525.

²Writing the KV cache newly generated during the decode phase can be overlapped with computation in the attention layer.