ASTRA-MoE: GPU-Augmenting In-Storage Acceleration for Long-Context Mixture-of-Expert Inference

Hyeonggyu Jeong
Hanyang University
Seoul, Republic of Korea
hgjeong753@hanyang.ac.kr

Inyoung Song
Hanyang University
Seoul, Republic of Korea
iysong0222@hanyang.ac.kr

Jinwoo Jeong
Hanyang University
Seoul, Republic of Korea
jwjeong@hanyang.ac.kr

Kyungmo Koo
Hanyang University
Seoul, Republic of Korea
kookyungmo@hanyang.ac.kr

Byungmin Ahn
Samsung Electronics
Seoul, Republic of Korea
bm.ahn@samsung.com

Dong-Min Shin
Samsung Electronics
Seoul, Republic of Korea
dm1229.shin@samsung.com

Yong Ho Song
Samsung Electronics
Seoul, Republic of Korea
yongho7.song@samsung.com

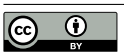
Jungwook Choi
Hanyang University
Seoul, Republic of Korea
choij@hanyang.ac.kr

Abstract

Large-scale Mixture-of-Experts (MoE) inference exposes the limits of GPU-alone systems: a single GPU lacks HBM for the terabyte-scale expert pool and pays 100–500× the compute time per CPU-DRAM or SSD weight transfer, while multi-GPU expert parallelism wastes HBM on static experts and stalls under skewed routing (*MoE Stall*). We argue that a small amount of compute inside the SSD can serve as a high-leverage *helper*: ASTRA-MoE augments a conventional SSD with only marginal additions (an edge-class Expert Processing Unit within standard SSD power/area budgets, plus a custom NVMe MoE command), keeping cold-expert weights in storage while the host exchanges only per-step activations and overlaps GPU and SSD execution at sub-batch granularity. Two co-designed optimizations: *Temporal Hot-Expert Replacement* (THER) ranks experts by accumulated routed-token count to map placement to GPU-vs-EPU critical-path cost rather than memory-access events; and *Tile-based Expert Processing* (TEP) combines a column-wise flash layout with out-of-order tile dispatch to keep the EPU pipelined despite ECC retries. Calibrated against our *OpenASTRA* FPGA prototype, ASTRA-MoE delivers up to **2.5× higher decoding throughput** than MoE-Infinity and **83%/34% lower per-token energy** versus single-GPU and the strongest multi-GPU baselines—from a near-conventional SSD rather than additional GPUs, establishing in-storage processing as a low-cost, energy-efficient path for MoE serving.

CCS Concepts

• **Computer systems organization → Heterogeneous (hybrid) systems**; • **Hardware → External storage**.



This work is licensed under a Creative Commons Attribution 4.0 International License.
PACT '26, Chicago, IL, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2915-7/2026/10
https://doi.org/10.1145/3838684.3846875

Keywords

Large Language Model Inference, Mixture-of-Experts, In-Storage Computing, SSD

ACM Reference Format:
Hyeonggyu Jeong, Inyoung Song, Jinwoo Jeong, Kyungmo Koo, Byungmin Ahn, Dong-Min Shin, Yong Ho Song, and Jungwook Choi. 2026. ASTRA-MoE: GPU-Augmenting In-Storage Acceleration for Long-Context Mixture-of-Expert Inference. In *International Conference on Parallel Architectures and Compilation Techniques (PACT '26), October 19–22, 2026, Chicago, IL, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3838684.3846875

1 Introduction

The Mixture-of-Experts (MoE) paradigm has become a cornerstone in the evolution of Large Language Models (LLMs), enabling an unprecedented expansion in model capacity without a proportional increase in computational cost [8, 43, 55, 56]. By partitioning dense Feed-Forward Network (FFN) layers into numerous smaller "expert" networks, MoE models sparsely activate only a fraction of parameters per input token. This sparse activation is particularly advantageous during the memory-bandwidth-bound decode stage, as it drastically reduces the volume of data that must be fetched to generate each token. However, this massive scalability introduces a severe memory capacity bottleneck. Even though only a few experts are active at any given moment, the entire terabyte-scale pool of expert parameters must remain continuously accessible to support dynamic routing, a requirement that far exceeds the High-Bandwidth Memory (HBM) limits of modern GPUs [10, 17, 24, 47].

Current GPU-alone systems are fundamentally inefficient for MoE inference, and the inefficiency becomes critically pronounced under long-context workloads. The first strategy, single-GPU data offloading, compensates for the GPU's limited HBM capacity by holding the expert pool in external CPU DRAM or NVMe SSDs and fetching weights on demand [23, 38, 40, 44, 47]. However, this approach is fundamentally crippled by the exorbitant latency of transferring massive expert weights over the PCIe bus at every decoding step—100–500× the corresponding GPU compute time. Recent cache-augmented variants retain frequently-reused experts

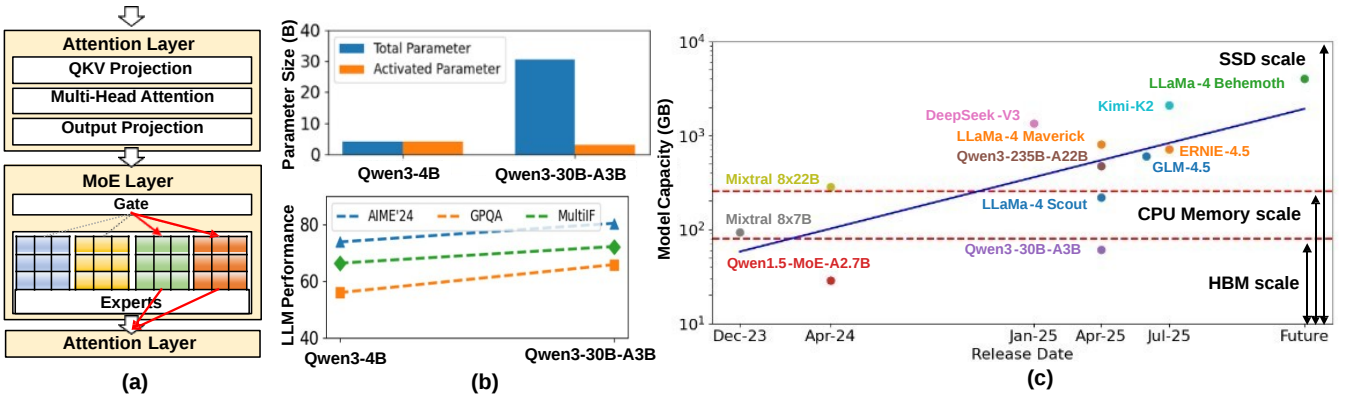


Figure 1: Characteristics of MoE LLM. (a) MoE Structure. (b) Dense vs MoE: Parameter size and LLM accuracy. (c) Trends in MoE Model Size.

in HBM to amortize this cost [28, 50, 52], yet they still execute every selected expert on the GPU after weight migration; once the multi-user active working set exceeds the GPU-resident cache, frequent expert swaps recover the original PCIe pathology.

The second strategy, multi-GPU expert parallelism, distributes the experts across a cluster's aggregate HBM [4, 39, 54]. While this avoids PCIe bottlenecks, the static expert weights permanently lock up a massive fraction of the distributed memory. As context lengths grow, the expanding Key-Value (KV) cache quickly exhausts the scarce remaining HBM, forcing systems to drastically shrink the decode batch size to avoid Out-Of-Memory (OOM) failures, thereby plummeting overall throughput. Furthermore, multi-GPU execution suffers from severe workload imbalance—termed *MoE Stall*—where highly skewed, input-dependent routing concentrates tokens on a small set of heavily-routed (*hot*) experts while the GPUs hosting sparsely-routed (*cold*) experts sit idle.

To unlock scalable long-context MoE inference, we argue that augmenting the GPU system with a small amount of compute placed inside the SSD—rather than replacing the GPU—can serve as a high-leverage *helper* that dramatically improves GPU efficiency on MoE workloads [16, 27, 37, 46]. This hot-GPU/cold-near-data execution principle has already been explored for MoE in near-memory systems such as MoNDE [22] and HD-MoE [15]; ASTRA-MoE instead realizes it as a storage-tier system built around a conventional NVMe SSD, under the flash, FTL, and ECC constraints that a near-memory design does not face. By keeping the massive expert parameter footprint inside the storage device and exchanging only lightweight per-step activations, the host GPUs can reclaim their HBM for KV cache expansion, enabling vastly larger batch sizes while the SSD-resident compute absorbs the memory-bound cold-expert tail. However, realizing in-storage MoE acceleration is severely hindered by the strict physical constraints of SSDs. Unlike standalone accelerators, storage devices operate under stringent thermal, power, and area budgets, resulting in a highly constrained embedded NPU and limited internal flash bandwidth. Existing in-storage LLM accelerators [27] target dense models and offload computations uniformly, ignoring MoE routing skew; under such MoE-blind offloading, hot experts saturate the constrained SSD-side compute and convert the storage device into a new system bottleneck. An effective design must therefore selectively retain heavily-loaded experts on the GPU while routing only the memory-bound cold tail to the SSD.

To overcome these fundamental challenges, we propose ASTRA-MoE, a GPU-augmenting hybrid architecture that respects routing-induced workload skew under strict SSD resource limits. Rather than provisioning additional GPUs or embedding a power-hungry accelerator within storage, ASTRA-MoE augments a conventional SSD with only marginal structural changes and an edge-class Expert Processing Unit (EPU) that fits within standard SSD energy budgets and preserves standard flash management and control mechanisms. By orchestrating a hybrid execution model that keeps heavily-routed hot experts on the GPU and routes only the memory-bound cold tail to the SSD, ASTRA-MoE demonstrates that a marginal hardware addition, when tightly integrated with existing storage behaviors, is sufficient to substantially improve GPU efficiency on MoE workloads. This paper makes the following key contributions:

- A novel hybrid inference architecture, ASTRA-MoE, built around ASTRA-SSD and a custom NVMe MoE command that exchanges only per-step activation pointers and tensor metadata across PCIe, allowing cold-expert weights to remain resident in the SSD and be consumed locally by the EPU. The host runtime overlaps the two paths at sub-batch granularity, so each decoding step pays close to the slower of GPU and SSD execution rather than their full sum; the overlap is not perfect, as synchronization and result-transfer overhead are still incurred, and SSD execution can enter the critical path if it exceeds the paired sub-batch's GPU work.
- Two co-designed optimizations: *Temporal Hot-Expert Replacement* (THER), a load-aware residency policy that keeps experts with the largest routed-token load on the GPU, prioritizing how many tokens each expert handles over how often it is selected; and *Tile-based Expert Processing* (TEP), which combines a column-wise flash layout with out-of-order tile dispatch to keep the EPU pipelined despite ECC retries and out-of-order page returns.

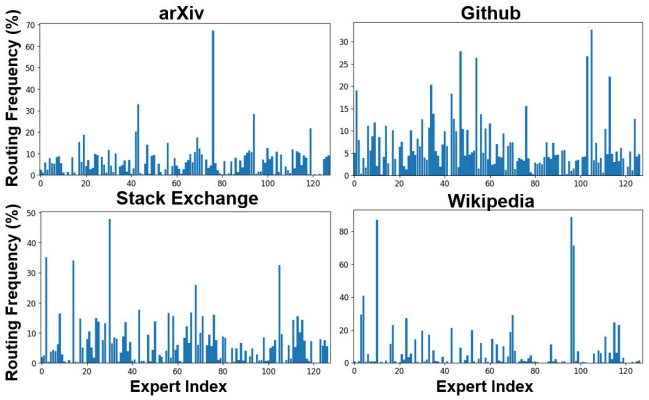


Figure 2: Domain-dependent expert skew across diverse datasets (Qwen3-30B-A3B, Layer #42). Routing frequency denotes the number of tokens routed to each expert, normalized by the total number of input tokens.

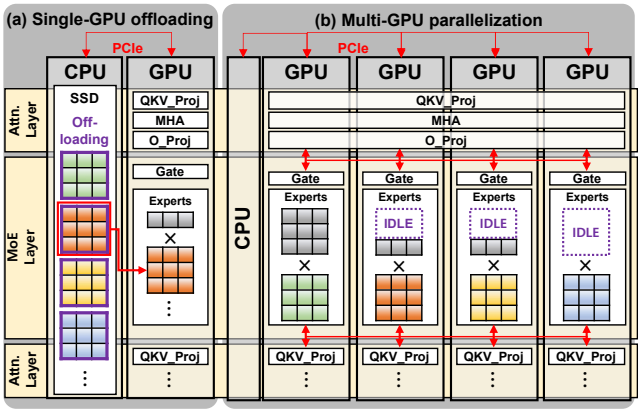


Figure 3: GPU system for MoE LLM Inference: (a) Single-GPU system with expert offloading, and (b) Multi-GPU system with expert parallelization.

- An evaluation calibrated against *OpenASTRA*, our FPGA prototype on the Cosmos+ platform, showing the throughput gap over baselines widening with context length. Ablation: THER alone already outperforms all baselines, and TEP further compounds these gains. Overall, ASTRA-MoE delivers up to **2.5**× higher decoding throughput (median **2.03**×) than MoE-Infinity and up to **83%/34%** lower per-token energy (median **59.9%/6.3%**) than single-GPU and the strongest multi-GPU baselines, while using a near-conventional SSD rather than additional GPUs, establishing in-storage processing as a low-cost, energy-efficient path for MoE serving.

2 Background

2.1 Scaling Mixture of Experts (MoE) LLMs

Mixture-of-Experts (MoE) models scale LLM capacity by replacing dense FFN layers with a router and many expert networks, while activating only a small top-K subset per token. This sparse activation keeps the active parameter size close to that of smaller dense models, but allows the total parameter pool to grow substantially, improving model capability without a proportional increase in per-token computation (Fig. 1(a)(b)).

This scalability creates a memory-capacity bottleneck during inference. Although each token activates only a few experts, the entire expert pool must remain accessible because routing is input-dependent. As shown in Fig. 1(c), recent MoE models already exceed single-GPU HBM and even CPU-memory scale, pushing the expert parameter pool into SSD-scale capacity. Thus, the central systems challenge is not only computing sparse experts efficiently, but also making a terabyte-scale expert pool continuously available under long-context serving.

2.2 Hot and Cold Experts for MoE Execution

MoE execution is highly skewed because routing assigns very different numbers of tokens to different experts. Within a decoding batch, a small subset of experts can receive many routed tokens (*hot experts*), while many others receive few or none (*cold experts*).

Table 1: MoE CPU-DRAM/SSD Offloading Latency

MoE Model	Qwen3-30B-A3B	Qwen3-235B-A22B	LLaMa4-Scout	LLaMa4-Maverick
Total Parameter	60GB	470GB	218GB	800GB
Active Parameter per Token	6GB	44GB	34GB	24GB
GPU Compute Latency	1.79ms	13.13ms	10.15ms	7.16ms
CPU Offloading Latency	0.187s	1.375s	1.063s	0.75s
SSD Offloading Latency	0.824s	6.05s	4.452s	3.185s

This skew is also domain-dependent: as shown in Fig. 2, experts that are frequently selected for one input domain may be rarely selected for another.

This hot/cold behavior implies that experts should not be treated uniformly. Hot experts expose enough token-level parallelism and reuse to execute efficiently on the GPU, whereas cold experts are fragmented, memory-bound, and often underutilize GPU resources. An efficient MoE system should therefore retain hot experts on the GPU while moving cold-expert execution to a lower-overhead near-data path.

2.3 Inefficiencies of Conventional GPU Systems for MoE Inference

Executing large MoE models on conventional GPU systems, particularly during the decode phase of autoregressive serving, introduces critical system-level inefficiencies. While sparse expert activation improves computational efficiency, it imposes severe memory capacity and data movement bottlenecks that current GPU-centric architectures struggle to resolve. These limitations manifest distinctly across single-GPU and multi-GPU deployments.

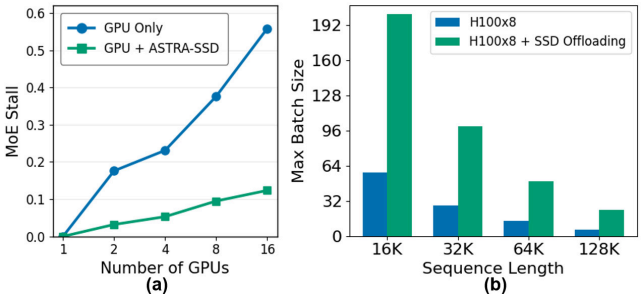


Figure 4: MoE inefficiency. (a) MoE stall caused by domain-dependent expert skew for Qwen3-30B-A3B on H100 GPUs across different input domains (batch size = 64). (b) Batch-size limitation in long-context inference.

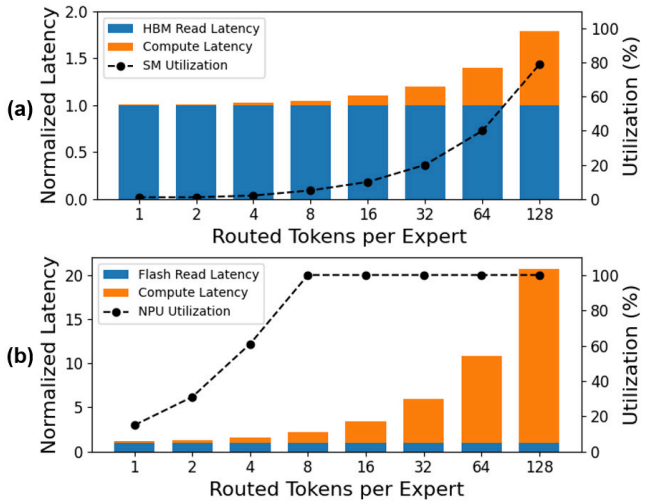


Figure 5: Latency breakdown and compute-core utilization across varying routed tokens per expert. (a) GPU-A100 (312 TFLOPS). (b) In-SSD EPU (2 TFLOPS). In each plot, latency is normalized to the single-expert memory-read latency of the corresponding device.

Single-GPU Inference: Data Offloading Bottleneck. In a single-GPU setup, the primary challenge is the GPU HBM's severe memory capacity constraint. As illustrated in Fig. 3(a), expert parameters must be offloaded to higher-capacity but significantly slower storage tiers, such as host CPU DRAM or NVMe SSDs, and fetched back to the GPU on demand to avoid Out-Of-Memory (OOM) errors when serving massive MoE models. While technologies like GPUDirect Storage (GDS) and GPUDirect RDMA (GDR) attempt to streamline this data movement [34, 35], the approach remains fundamentally crippled by the limited bandwidth of the host–GPU PCIe I/O path. Furthermore, the dynamic, input-dependent routing of MoE models makes effective weight prefetching difficult. As quantified in Table 1, this data offloading strategy introduces a massive performance penalty: transferring expert weights from host memory or storage incurs a latency 100–500× higher than the actual GPU computation time. This data transfer bottleneck completely dominates the inference execution, rendering single-GPU offloading highly inefficient for high-throughput serving.

Multi-GPU Inference: Expert Parallelism Inefficiencies. To overcome single-GPU capacity limits, large MoE models are commonly distributed across GPUs via Expert Parallelism (EP), which partitions experts across devices and routes tokens through All-to-All communication. EP is preferred over Tensor Parallelism (TP) for MoE: TP relies on fine-grained All-Reduce to merge partial tensors and scales efficiently to only a few GPUs, whereas EP avoids per-tensor collectives by simply routing each token to the device that owns its selected experts [39]. Fig. 3(b) illustrates this layout: each GPU is assigned a disjoint subset of experts, and tokens are exchanged across GPUs via All-to-All communication before and after each MoE layer's expert computation. Yet EP introduces two critical inefficiencies that worsen at scale (Fig. 4):

- **MoE Stall from Expert Imbalance (Fig. 4(a)):** Skewed routing concentrates tokens on the GPUs that host hot experts; GPUs holding only cold experts receive almost no work and sit idle. Because the next layer cannot proceed until every GPU finishes, this waiting directly manifests as *MoE Stall*, defined as $\textit{MoE Stall} = \sum_g T^{\text{MoE}}_{g,\text{idle}} / (T^{\text{MoE}} \cdot N_{\text{GPU}})$, where $T^{\text{MoE}}_{g,\text{idle}}$ is GPU g's idle time within the MoE layer, T^{MoE} is the layer latency, and N_{GPU} is the GPU count. Fig. 4(a) shows the penalty grows with N_{GPU}, i.e., adding GPUs makes the imbalance worse, not better.
- **Limited Batch-Size Scaling (Fig. 4(b)):** Even after pooling HBM across GPUs, the static expert footprint occupies a disproportionate fraction of aggregate memory, leaving little room for the Key-Value (KV) cache, which scales linearly with batch size and context length. Long contexts therefore force aggressive batch-size shrinkage to avoid OOM, collapsing GPU utilization just when high throughput matters most.

In short, GPU-only architectures are fundamentally mismatched to large-scale MoE inference. Single-GPU offloading is dominated by repeated PCIe weight transfers, while multi-GPU expert parallelism wastes HBM on static experts and suffers from MoE Stall and KV-cache starvation. Adding more GPUs therefore does not directly solve the MoE serving problem; the expert footprint must be moved out of HBM, and a small amount of compute should be placed where the experts already reside.

3 Opportunities and Challenges of In-Storage MoE Acceleration

3.1 Opportunities of In-Storage MoE Acceleration

In-storage MoE acceleration improves GPU efficiency by placing lightweight compute next to NAND-resident expert weights. Instead of storing the full expert pool in GPU HBM or repeatedly transferring expert weights over PCIe, the SSD keeps cold-expert weights local and exchanges only per-step activations with the host. This reclaims HBM for KV cache, enables larger decode batches, and removes the rigid expert partitioning that causes MoE Stall in multi-GPU EP.

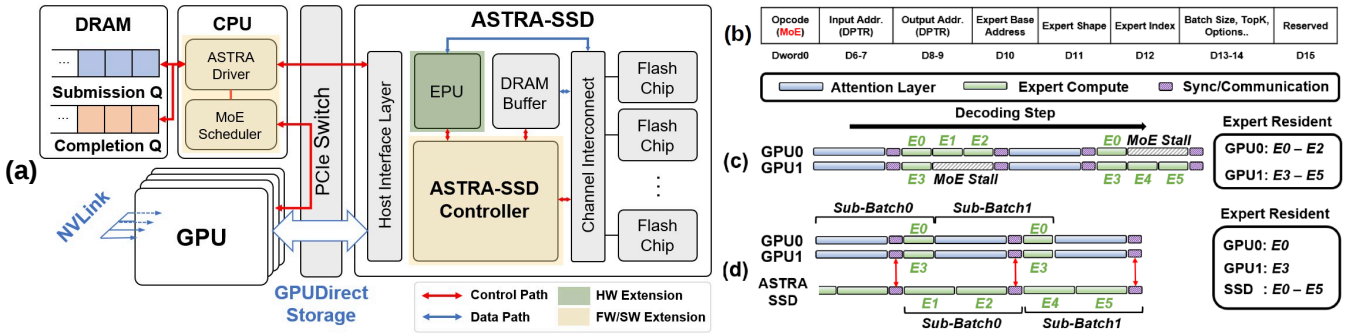


Figure 6: System integration of ASTRA-MoE. (a) CPU–GPU–ASTRA-SSD system overview, with HW and FW/SW extensions over a conventional SSD marked accordingly. (b) NVMe MoE command format and metadata fields. (c) Conventional multi-GPU MoE execution pipeline. (d) Multi-GPU MoE execution pipeline with ASTRA-SSD.

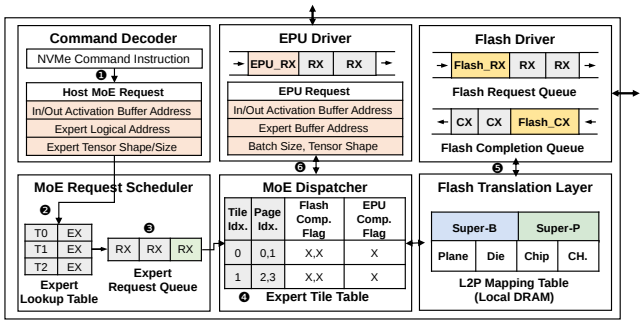


Figure 7: ASTRA-SSD controller architecture

This opportunity is practical because cold experts do not require GPU-class compute. As shown in Fig. 5, experts with few routed tokens are dominated by memory-read latency on the GPU and exhibit low SM utilization, while a lightweight in-SSD EPU can process this fragmented workload efficiently. The EPU only needs to absorb the memory-bound cold tail; hot experts with high routed-token counts remain on the GPU, where their parallelism and reuse are better exploited.

A key advantage is that this design requires only marginal hardware additions to an existing SSD. ASTRA-SSD adds an edge-class EPU and lightweight command/scheduling logic, while reusing the SSD's flash channels, FTL, DRAM buffer, and controller infrastructure. Thus, in-storage MoE acceleration acts as a GPU helper rather than a GPU replacement, improving throughput and energy efficiency without provisioning additional GPU-class compute.

3.2 Challenges

Although in-storage MoE acceleration presents promising opportunities, realizing it in practice requires overcoming three critical challenges stemming from storage devices' physical constraints:

- **Architectural Integration under Strict Resource Limits.** Unlike standalone accelerators, an SSD operates under stringent power and area constraints. Consequently, EPU cannot be a massive, high-performance processor; it must be restricted to an edge-class accelerator [13, 32, 49]. The primary architectural challenge is to seamlessly integrate this highly constrained compute unit into the existing host–SSD interface (e.g., NVMe) and standard flash control mechanisms without disrupting normal storage operations.
- **Dynamic GPU-SSD Workload Balancing.** Because the EPU has much lower compute capability than a GPU, indiscriminately offloading all experts would overload the SSD path. The system must dynamically keep hot, compute-heavy experts on the GPU while assigning memory-bound cold experts to the SSD, despite input-dependent routing changes.
- **Accelerator-Level Performance Optimization.** Scaling EPU FLOPs is expensive under SSD constraints, so performance must come from reducing internal inefficiencies. The SSD must overlap flash reads with EPU execution and tolerate non-deterministic flash completions, including ECC read retries, to avoid wasting the limited EPU capability.

4 Methodology: ASTRA-MoE

To overcome prior limitations, we propose **A**ccelerated **STo**R**a**ge for **A**ggregated MoE Compute Offloading System (ASTRA-MoE), a hybrid GPU-and-in-storage computing architecture that avoids data-movement bottlenecks in MoE inference. We first outline the system architecture across the host, GPU, and SSD, followed by our system-level and accelerator-level co-optimizations.

4.1 System Architecture

The end-to-end integration of ASTRA-MoE within a standard CPU-GPU-SSD topology relies on a specialized host-storage interface to orchestrate offloaded computations (Fig. 6(a)). Rather than transferring massive expert parameters over the PCIe bus, ASTRA-MoE exchanges only the lightweight per-step input and output activations of the MoE layers. To seamlessly support this without breaking standard storage protocols, we introduce the NVMe MoE command (Fig. 6(b)). This command reuses the standard data pointer (DPTR) mechanism to reference activation buffers in host or GPU memory, while embedding essential execution metadata—such as expert logical addresses, tensor shapes, batch sizes, and top-K parameters—directly into the command fields. This design enables the host's lightweight MoE scheduler to dispatch execution requests

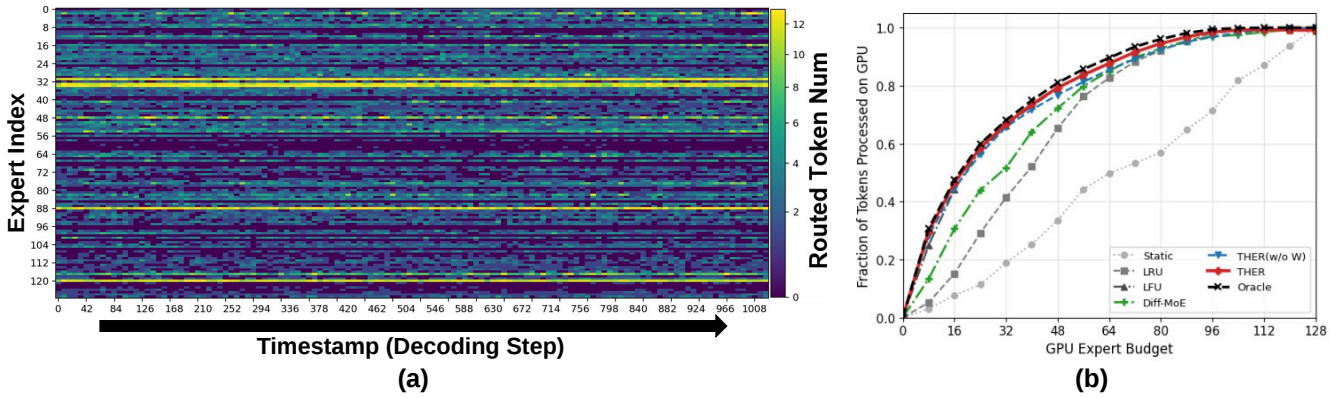


Figure 8: (a) Temporal locality of selected experts across decoding steps in layer 16 of Qwen3-235B-A22B on RedPajama-1T inputs (batch size = 32). (b) GPU-processed token fraction across expert residency policies during batched inference with a batch size of 128. The y-axis denotes the fraction of routed tokens in each decoding step that are assigned to GPU-resident experts. THER w/o W accumulates routed-token counts over the entire execution history without temporal windowing. Diff-MoE denotes its adapted residency policy under the same GPU expert-memory budget. Oracle selects the top-B experts using perfect knowledge of routed-token loads over the upcoming F-step replacement interval while obeying the same replacement period as THER.

per MoE-layer invocation while abstracting away the SSD's internal scheduling complexities.

To maximize decoding throughput, ASTRA-MoE employs a hybrid GPU-ASTRA execution pipeline that aggressively overlaps host and storage computations (Fig. 6(d)). This targets the conventional multi-GPU pipeline of Fig. 6(c), where all routed experts execute on GPUs and a GPU holding only cold experts must sit idle until every GPU finishes its share of the layer, producing the *MoE Stall* introduced in Section 2. ASTRA-MoE instead routes that cold-expert share of the sub-batch to the SSD, turning idle GPU wait time into SSD execution overlapped with the GPU's own resident-expert work. During autoregressive generation, a selected subset of "hot" experts remains resident in the GPU High-Bandwidth Memory (HBM), while the full expert pool resides in the ASTRA-SSD. At each decoding step, the input batch is divided into sub-batches. As the GPU executes attention and processes its resident experts for the current sub-batch, the ASTRA-SSD simultaneously computes the non-resident experts for another sub-batch. The host runtime orchestrates this overlap via asynchronous NVMe submission/completion queues (SQ/CQ) and CUDA events, reducing the exposed SSD execution latency by overlapping it with ongoing GPU operations.

This overlap is not assumed to be perfect: result transfer and synchronization are counted as part of the communication latency, and if SSD-side execution exceeds the available GPU-side work for the paired sub-batch, the SSD path becomes part of the critical path. Within a given sub-batch, the next layer still cannot proceed until both its GPU- and SSD-computed expert outputs are merged. Therefore, balancing computation between the GPU and SSD must be carefully considered; the THER policy described in Section 4.2 is designed to prevent the SSD path from becoming the critical path.

Inside the device (Fig. 7), each request flows through a small added pipeline: the *Command Decoder* parses the NVMe MoE command, the *MoE Request Scheduler* groups work by expert index using an Expert Lookup Table, and the *MoE Dispatcher* coordinates the Flash Driver and FTL to stream expert pages into Local DRAM and issues assembled tiles to the embedded *EPU*, which performs the expert GEMM and writes outputs back for host DMA. Importantly, the FTL, Flash Driver, channels, and DRAM controller are inherited unchanged from a conventional SSD; only the Decoder, Scheduler, Dispatcher, and EPU are added—marginal logic that fits within the device's standard power and area envelope (Fig. 7).

Although expert identity varies unpredictably across decoding steps, each selected expert (e.g., ~36 MB for Qwen3-235B-A22B) is accessed as a large, page-granular object under TEP (§4.3), rather than through many scattered fine-grained reads. Expert weights are read-only during inference, so the expert-read path itself generates no write traffic and does not induce write-triggered garbage collection. The controller-side DRAM buffer is used only for transient tile staging rather than as a persistent expert cache.

4.2 System-Level Optimization: GPU-ASTRA Expert Partitioning

ASTRA-MoE must decide which experts remain on the GPU and which are served by the ASTRA-SSD. Offloading too aggressively overloads the constrained EPU, while updating GPU-resident experts too frequently incurs SSD-to-GPU migration overhead. As shown in Fig. 8(a), several experts remain repeatedly active across consecutive decoding steps, exhibiting clear temporal persistence in routed-token load. THER exploits this temporal locality: recent routing history is predictive enough that the GPU-resident set can be refreshed periodically rather than at every decoding step.

THER differs from conventional expert-residency policies in its optimization objective. LRU/LFU-style policies treat expert selection as a binary access event, whereas in batched MoE serving the SSD-side execution cost depends on the number of tokens routed to each expert. Similarly, conventional expert-offloading systems such

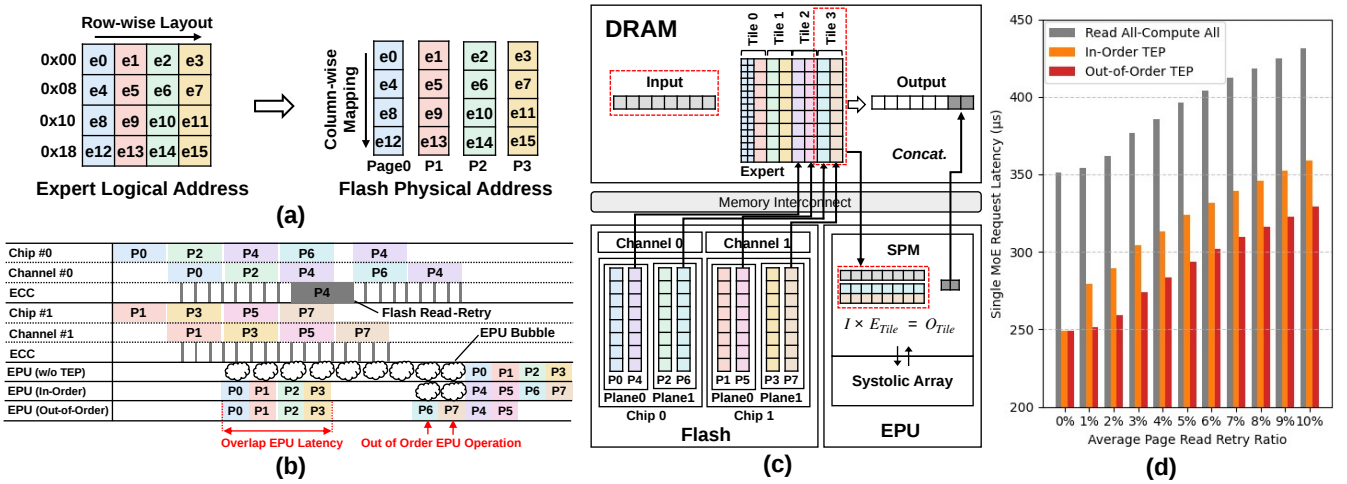


Figure 9: Tile-based Expert Processing. (a) Logical-to-physical layout transformation of expert weights. (b) EPU bubbles in conventional transaction-based, in-order execution under out-of-order flash read completions caused by flash-level parallelism and ECC read retries. (c) TEP architecture. (d) Performance improvement over the Read-Retry ratio.

as Diff-MoE [28] primarily use expert reuse and prediction to reduce costly expert-weight migration to the GPU. In ASTRA-MoE, however, a non-resident expert remains in storage and is executed on the resource-constrained SSD-side EPU; therefore, expert residency also determines how routed computation is partitioned between the GPU and SSD.

THER therefore accumulates per-expert routed-token counts over a history window W, updates residents every replacement period F, and keeps the top-B experts with the largest accumulated token load on the GPU (Algorithm 1). This prevents heavily routed experts from falling onto the SSD path while allowing cold experts to be processed near the data. To isolate the effect of temporal windowing, we additionally evaluate THER w/o W, which uses the same routed-token-count ranking but accumulates counts over the entire execution history. THER instead retains only the most recent W-step routing history, allowing the resident set to adapt to temporal changes in expert load. Fig. 8(b) shows that THER processes a larger fraction of routed tokens on the GPU than Static, LRU, LFU, THER w/o W, and the adapted Diff-MoE, and approaches the Oracle. The result confirms that token-load-aware residency captures near-future hotness without offline knowledge, improving GPU-side parallelism while reducing the chance that the SSD-resident EPU becomes the critical path.

4.3 Accelerator-Level Optimization: Tile-Based Expert Processing

Even with correct GPU–SSD partitioning, ASTRA-SSD can stall if flash reads and EPU execution are serialized. A conventional row-wise layout forces the EPU to wait until a large expert block is fetched, even though the EPU consumes weights as column-aligned output tiles. TEP remaps expert weights into a column-wise physical layout so that each tile becomes page-contiguous and can be consumed as soon as its pages arrive in the local buffer (Fig. 9(a)).

Beyond this layout transform, TEP addresses two distinct sources of EPU idle time: coarse-grained fetch-before-compute serialization, and head-of-line blocking from non-deterministic flash-page completion. TEP therefore uses tile-level readiness tracking to overlap flash reads with EPU execution and tolerate non-deterministic flash completions. As Fig. 9(b) notes, this non-determinism has two distinct origins: flash-level parallelism, where a tile's pages are striped across independent channels, chips, and planes that complete asynchronously even without error, and ECC read retries, which delay an individual page beyond its already-variable completion time. Flash-level parallelism alone routinely reorders page arrivals, so this readiness tracking matters even in the common, retry-free case, not only to tolerate the comparatively rarer retry events. For each expert request, the MoE Dispatcher instantiates an *Expert Tile Table* entry (Fig. 7) that records per-tile flash/EPU completion flags; the Dispatcher then issues any tile whose data is ready, rather than enforcing flash-issue order. This out-of-order dispatch prevents ECC read retries from blocking unrelated ready tiles (Fig. 9(b)(c)). As Fig. 9(d) shows, out-of-order TEP keeps latency nearly flat as read-retry ratio increases, converting the SSD path from a fetch-then-compute sequence into a continuously pipelined regime.

5 Experiments

5.1 Experimental Settings

MoE Workloads. We evaluate ASTRA-MoE on three MoE models spanning single-GPU, multi-GPU, and frontier-scale settings: Qwen3-30B-A3B, LLaMa4-Scout, and Qwen3-235B-A22B (Table 3). Requests are sampled from six domains—arXiv, WikiText, NarrativeQA, GovReport, GitHub, and Stack Exchange—to capture realistic routing skew. Each model uses the minimum GPU count that fits its parameters in HBM (1/4/8 GPUs), and we sweep 32K–128K context lengths with the largest decode batch that fits after the

Algorithm 1 Temporal Hot-Expert Replacement (THER), executed once per decoding step t

Require: Hyperparameters: GPU expert budget B, history window W, replacement period F

Persistent state (per layer l, kept across decoding steps):
$\mathcal{E}_l$: universe of experts in layer l, $|\mathcal{E}_l| = N$
H_l: sequence of the last $\le W$ routed-token multisets
S_l: hotness multiset over $\mathcal{E}_l$; invariant $S_l = \biguplus_{c \in H_l} c$
$R_l \subseteq \mathcal{E}_l$: GPU-resident expert set, $|R_l| = B$

Input at step t: routed-token multiset $c_l^{(t)}$ over $\mathcal{E}_l$, where the multiplicity of expert e in $c_l^{(t)}$ is the number of tokens routed to e in layer l

for all layers l **do**
  *// **Phase 1: update hotness multiset** ($O(N)$ **per step**)*
  if $|H_l| = W$ **then**
    $c_{\text{old}} \leftarrow$ pop front of H_l
    $S_l \leftarrow S_l \setminus c_{\text{old}}$ ▷ multiset difference
  end if
  push $c_l^{(t)}$ to back of H_l
  $S_l \leftarrow S_l \uplus c_l^{(t)}$ ▷ multiset sum
  *// **Phase 2: refresh GPU residents every** F **steps***
  if $t \bmod F = 0$ **then**
    $R_l^{\text{new}} \leftarrow \text{TopB}(S_l, B)$ ▷ B experts with largest multiplicity in S_l
    for all $e \in R_l \setminus R_l^{\text{new}}$ **do**
      evict expert e from GPU
    end for
    for all $e \in R_l^{\text{new}} \setminus R_l$ **do**
      stream expert e weights from SSD to GPU
    end for
    $R_l \leftarrow R_l^{\text{new}}$
  end if
end for

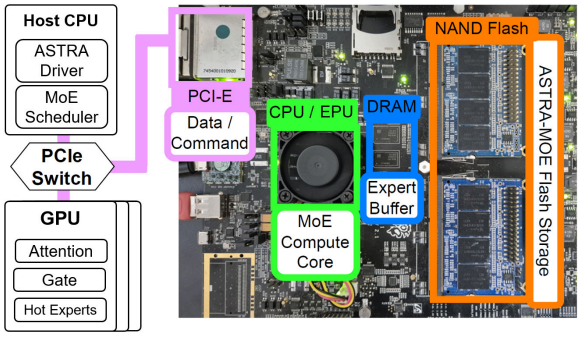


Figure 10: FPGA-based validation of OpenASTRA.

resident KV cache. All model parameters and expert computation use BF16 precision.

Baselines. We compare against DeepSpeed-Zero-Inference, vLLM expert parallelism, MoE-Infinity, and AiF [2, 26, 27, 50]. DeepSpeed represents single-GPU expert offloading, vLLM represents the GPU-only multi-GPU ceiling, MoE-Infinity represents state-of-the-art GPU+SSD expert caching without in-storage compute, and AiF represents dense-model in-flash compute offloading. For fairness, AiF is scaled to the same compute and flash bandwidth as ASTRA-SSD. Together, these baselines separate the effects of GPU-only execution, weight caching, uniform in-storage offloading, and ASTRA-MoE's MoE-aware SSD-controller-level execution.

Table 2: ASTRA-SSD System Configuration.

Specification	Detail	Bandwidth (Latency)	Capacity
GPU	A100-80GB HBM2e	2,039 GB/s	80 GB
	H100-80GB HBM3	3,350 GB/s	80 GB
GPU Interconnect	NVLink (A100)	600 GB/s	–
	NVLink (H100)	900 GB/s	–
Host Interface	PCIe 5.0 × 4 lanes	32 GB/s	–
DRAM	Samsung LPDDR5X [42] × 4	273.1 GB/s	16 GB
Flash Memory	Samsung TLC NAND [19]	34 μs (tR)	16 TB
Flash Channels*	ONFI 6.0 [36] × 32/128 channels	153.6/614.4 GB/s	–
EPU SPM	Double-buffered SRAM	512 GB/s	300 KB
EPU Core	[4×256] Systolic Array @ 1 GHz	2 TFLOPs	–
ECC Engine	4 KiB LDPC	100 ns/iter	–

*We use 32 flash channels for the single-GPU configuration and 128 channels for the 4- and 8-GPU configurations. Aggregate flash bandwidth scales linearly with channel count ($B_{\text{flash}} \propto N_{\text{ch}}$).

Table 3: MoE LLM Configurations.

MoE Model	Qwen3-30B-A3B	Qwen3-235B-A22B	LLaMa4 Scout
Model Parameter Size	30B	235B	109B
Active Parameters	3B	22B	17B
Number of Experts	128	128	16
Top-K Experts	8	8	1
Max Seq Length	256K	256K	10M

ASTRA-SSD Configuration. The EPU specifications in Table 2 are deliberately edge-class: a 2 TFLOPs 4×256 systolic array—roughly 150× below an A100's 312 TFLOPs and on par with mobile NPUs such as the Coral Edge TPU and NVIDIA Jetson [13, 32]—fitting within a conventional SSD's power and area envelope. The gains reported below therefore stem from the orchestration policies (THER, TEP) rather than from provisioning GPU-class compute inside storage.

FPGA Prototype and Performance Model. ASTRA-MoE requires SSD-internal coordination among flash access, DRAM buffering, EPU scheduling, and NVMe command/completion handling, which off-the-shelf simulators cannot validate end-to-end. We therefore built *OpenASTRA*, a functional FPGA prototype on the Cosmos+ OpenSSD platform [25] (Fig. 10), and use its cycle-level control overheads to calibrate MQSim [48] for SSD timing and ScaleSim [41] for EPU execution. Since the prototype runs below production

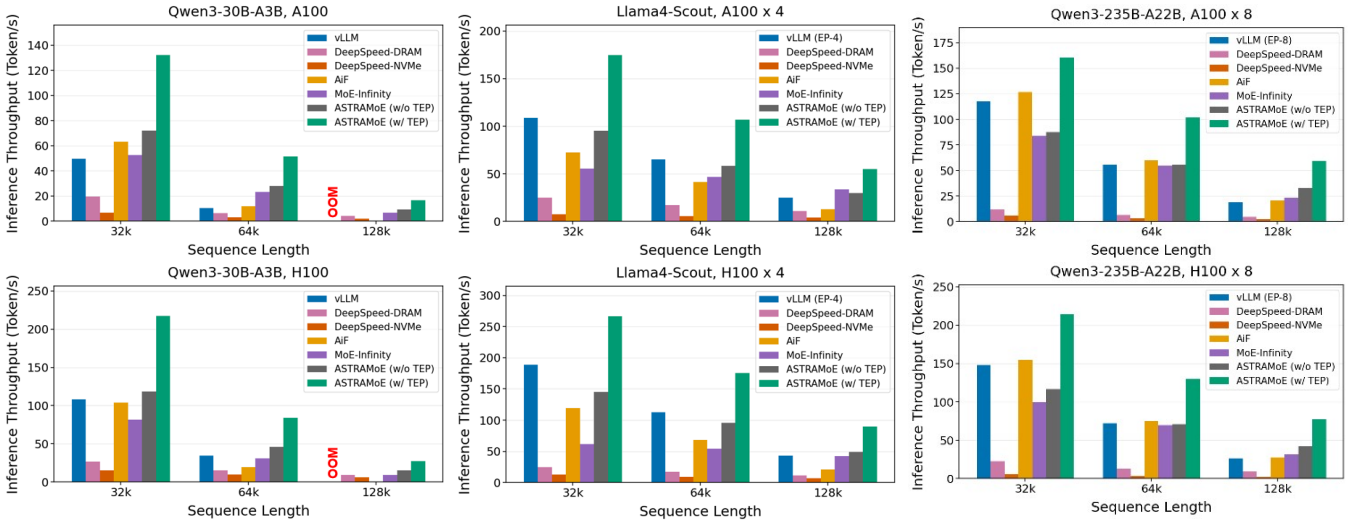


Figure 11: End-to-End decoding throughput across diverse MoE models under varying sequence lengths and GPU configurations. Each sequence length is composed of equal input and output token lengths. ASTRA-MoE sustains substantially larger batch sizes than the GPU-only baseline. At 32K/64K/128K, the batch sizes for ASTRA-MoE versus GPU-only are 24/13/6 vs. 6/3/1 for Qwen3-30B-A3B (1 GPU), 48/24/12 vs. 16/8/4 for LLaMa4-Scout (4 GPUs), and 68/42/22 vs. 26/12/6 for Qwen3-235B-A22B (8 GPUs), respectively.

clock and channel parallelism, only frequency and hardware parallelism are scaled to the target ASTRA-SSD configuration. Host-side attention, communication, and synchronization are taken from Nsight Systems [33] and Vidur [1] traces on real A100/H100 systems and are retained without re-optimization for ASTRA-MoE's presence. Our end-to-end evaluation replaces only the SSD-offloaded expert-execution segments with calibrated ASTRA-SSD latencies, making the methodology prototype-calibrated, trace-driven, and conservative: the reported end-to-end gains do not rely on any assumed host-side speedup, consistent with recent accelerator and disaggregated-system studies [37].

5.2 Experimental Results

5.2.1 End-to-End Throughput under Long Context. Fig. 11 reports decoding throughput across three MoE models, sweeping context length from 32K to 128K with the largest decode batch that fits in HBM after the resident KV cache. ASTRA-MoE outperforms all baselines across all operating points, and the gap widens with context length, where HBM pressure makes selective compute migration most valuable. As the per-request KV cache grows approximately linearly with context, the HBM reclaimed by keeping cold-expert weights off the GPU helps preserve a larger feasible decode batch and delays the batch-size reduction caused by KV-cache pressure.

The baselines exhibit four distinct failure modes. *vLLM* avoids expert-weight offloading overhead when HBM pressure is low, but becomes increasingly capacity-constrained as the KV cache grows with context length; for Qwen3-30B-A3B, it reaches OOM at 128K. *DeepSpeed-Zero* is the inverse: a flat, low throughput dictated by per-step PCIe weight transfer, immune to context but never competitive. *MoE-Infinity* caches frequently-used experts to defer this cost, but in multi-user batches the active working set exceeds GPU-resident capacity and residual swaps recover DeepSpeed's pathology. *AiF* does not consistently outperform vLLM across the evaluated configurations: MoE-blind uniform offloading can route heavily loaded experts into the constrained in-flash path, turning the SSD into a bottleneck. In every case, GPU-only and uniform-offload designs leave the GPU stalled on data movement or imbalance—the inefficiency is structural, not a tuning issue.

To isolate the contribution of each ASTRA-MoE technique, the plot also reports ASTRA-MoE without TEP, since disabling TEP removes Flash–EPU overlap and forces the SSD-side path to serialize flash reads with EPU compute. Even in this stripped configuration, ASTRA-MoE-w/o-TEP still outperforms every baseline across the full context sweep, demonstrating that THER's load-aware residency alone is sufficient to beat the cache-based and uniform-offload alternatives: mapping placement to per-expert token load both spares the EPU from heavy hot-expert work and keeps the GPU productively occupied. In addition to THER's placement gains, TEP delivers further throughput improvements at every operating point by overlapping flash reads with EPU computation, thereby converting SSD-side serialization into a continuously pipelined regime. The two techniques compound — THER establishes the placement decision that makes offloading viable, and TEP extracts additional throughput from that placement. Overall, ASTRA-MoE achieves 1.9–2.5× higher decoding throughput than MoE-Infinity, a state-of-the-art MoE offloading baseline (median 2.03× across all evaluated context lengths and models; the 2.5× peak occurs for Qwen3-235B-A22B at 128K on 8×A100), by combining THER's load-aware expert placement with TEP's Flash–EPU overlap.

5.2.2 GPU Expert Budget Trade-off. Fig. 12 sweeps the GPU expert budget across the 128 experts per layer of Qwen3-235B-A22B at 64K context on 8×A100, with stacked bars decomposing per-step latency

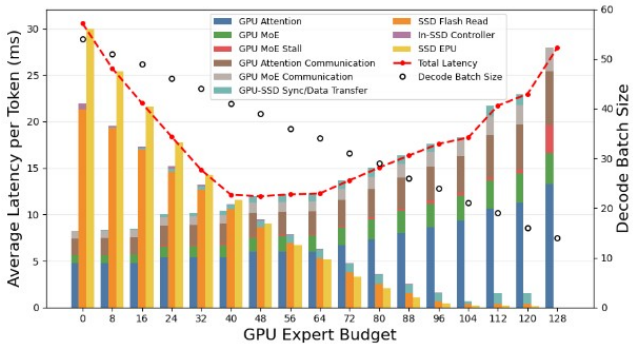


Figure 12: GPU–SSD latency breakdown across different GPU expert budgets for Qwen3-235B-A22B with a 64K sequence length on an 8×A100 GPU system and a 128-channel ASTRA-SSD. For each GPU expert budget, the three stacked bars denote the GPU-side latency, SSD-side latency, and SSD-EPU latency, respectively. The red dashed line indicates the resulting total per-token latency after accounting for TEP-enabled flash–EPU overlap. Open circles (right axis) report the batch size at each GPU expert budget.

into GPU-side and SSD-side components and the red dashed line reporting total per-token latency after GPU–SSD and Flash–EPU overlap.

Total latency follows a clear U-shape. At small budgets the GPU-side bar is short but the SSD-side bar is tall: the EPU is forced to serve heavily-loaded experts it is not built for. At large budgets the picture inverts—the SSD-side bar shrinks, but cold weights crowd out the KV cache, the batch shrinks, and GPU-side imbalance returns as GPUs hosting rarely-routed experts idle. The minimum sits in the interior, where neither path is saturated. The dashed total line stays consistently below the sum of the two bars: GPU and SSD execute in pipeline across sub-batches and Flash/EPU overlap via TEP, so the system pays only the slower stack. The budget is therefore a key control knob, and the optimum is the crossover at which the GPU and SSD paths balance.

5.2.3 THER Sensitivity to Update Granularity. Fig. 13 analyzes THER's sensitivity to F and W for Qwen3-235B-A22B at 64K sequence length. Aggressive updates track hotness changes quickly and reduce the chance that newly hot experts remain on the SSD path, but they also increase SSD-to-GPU expert replacement overhead. Conversely, conservative updates reduce replacement activity but can rely on stale hotness history. As shown in Fig. 13(a), conservative settings such as $(64, 1024)$ and $(256, 256)$ shift SSD-side MoE latency toward the tail because newly hot experts may remain on the SSD path.

Fig. 13(b) shows that the best throughput appears at a middle-ground setting. In this evaluated setup, $(64, 256)$ achieves the highest throughput of 104.0 tokens/s by balancing expert replacement overhead against stale-residency-induced SSD tail latency. Thus, F and W should be selected based on the target hardware and model configuration rather than fixed globally.

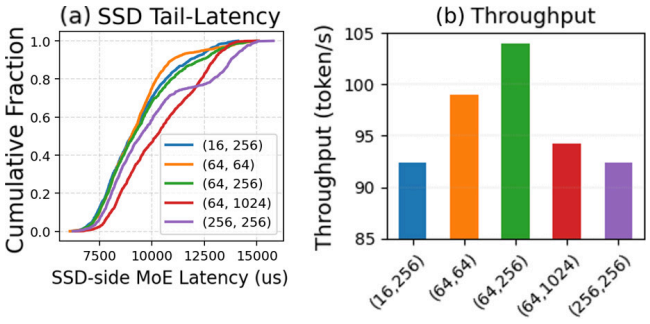


Figure 13: Sensitivity of THER to the replacement period and history window for Qwen3-235B-A22B at 64K sequence length. Each configuration is denoted by (F, W), where F is the replacement period and W is the history-window size. (a) CDF of SSD-side MoE latency. (b) End-to-end throughput. The middle-ground configuration $(64, 256)$ achieves the best throughput by balancing expert replacement overhead and SSD-side tail latency from stale residency decisions.

5.2.4 NVMe MoE Command. Fig. 14 compares GPU–SSD communication latency per decoding step at batch size 64, with and without the NVMe MoE command. Without it, the system migrates cold-expert weights from SSD to GPU at every step — each cold expert occupies tens of MB — so PCIe weight transfer dominates the communication time. With the command, only activation pointers and tensor metadata cross the interface; cold-expert weights stay resident in the ASTRA-SSD and are consumed locally by the EPU. The per-step communication latency therefore drops by the bulk of the weight-transfer term, leaving only synchronization and the much smaller activation traffic. Because this saving repeats every decoding step, it compounds across long-context generation.

5.2.5 Expert Residency Policy. Fig. 15 demonstrates that token-load-aware expert replacement is essential for achieving high throughput in ASTRA-MoE. Static performs poorly because a fixed GPU-resident set cannot follow the temporal shift of hot experts during decoding. LRU and LFU provide higher throughput by exploiting expert reuse, but their decisions are still based only on memory-access recency or frequency. Thus, they may prioritize experts that are accessed often but receive only a small number of routed tokens, while failing to prioritize less frequently accessed but heavily loaded experts. THER avoids this limitation by using the accumulated routed-token count as the hotness score, so experts who contribute more computation to the current batched workload are more likely to remain on the GPU. This directly reduces SSD-side EPU pressure and improves GPU-side token parallelism, leading THER to outperform LRU and LFU across both evaluated MoE models.

5.2.6 Sensitivity to EPU and Flash Performance. ASTRA-MoE is not narrowly tuned to a single EPU or flash configuration. Table 4 sweeps EPU throughput, flash bandwidth, and flash read latency for Qwen3-30B-A3B at a 32K sequence length on H100, relative to the default 2-TFLOPS EPU and nominal flash timing used elsewhere in this paper. Reducing EPU throughput from 2 to 1 TFLOPS decreases

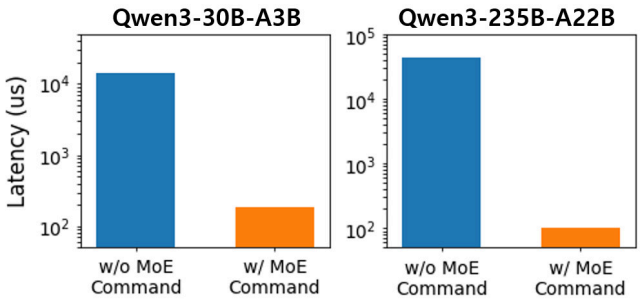


Figure 14: GPU–SSD communication latency per decoding step with and without the NVMe MoE command at batch size 64. The latency includes GPU–SSD synchronization and data transfer.

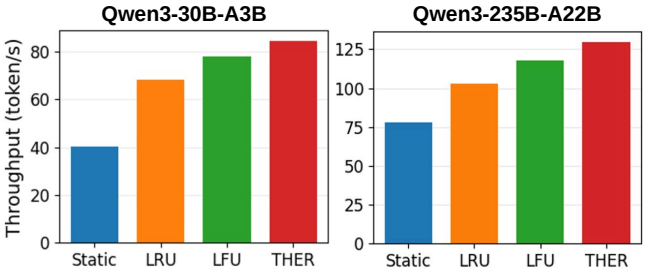


Figure 15: Inference throughput comparison across expert residency policies at sequence length 64K.

throughput by only 6.2%, while a further reduction to 0.5 TFLOPS causes a 37.7% drop; increasing EPU throughput beyond 2 TFLOPS provides no additional benefit, confirming that the EPU is not the bottleneck at our default configuration. Similarly, reducing flash bandwidth to 0.75× causes only a 0.6% loss, while 0.5× bandwidth reduces throughput by 9.7%. Increasing flash read latency to 1.5× and 2× reduces throughput by 7.4% and 19.4%, respectively. ASTRA-MoE therefore tolerates moderate EPU and flash variation, but performance degrades predictably once either resource becomes limiting.

5.2.7 Energy Efficiency. Fig. 16 reports the normalized per-token energy breakdowns for single-GPU and multi-GPU inference. We estimate component-level energy using execution traces and established energy and power parameters for GPU, NPU, DRAM, PCIe data movement, and NAND flash [5–7, 20, 49]. In the single-GPU setting, ASTRA-MoE reduces energy by up to 82.8% (median 59.9% across evaluated models) over DeepSpeed-NVMe by avoiding repeated expert-weight transfers over PCIe and executing cold experts near the data inside the ASTRA-SSD. DeepSpeed-NVMe spends a large fraction of its energy not only on SSD flash reads and PCIe movement, but also on GPU-HBM writes, since fetched expert weights must be repeatedly materialized in GPU memory before execution. In contrast, ASTRA-MoE keeps PCIe energy nearly negligible and avoids repeated GPU-HBM weight writes by keeping expert weights inside the ASTRA-SSD and exchanging only

Table 4: Sensitivity of ASTRA-MoE throughput to EPU compute and flash performance for Qwen3-30B-A3B at 32K sequence length on H100.

Parameter	Setting	Tput (tok/s)	Δ
EPU Throughput	0.5 TFLOPS	134.54	−37.7%
	1.0 TFLOPS	202.56	−6.2%
	2.0 TFLOPS (default)	215.95	—
	4.0 TFLOPS	215.95	—
Flash Bandwidth	0.5×	195.00	−9.7%
	0.75×	214.65	−0.6%
	1.0× (default)	215.95	—
Flash Read Latency	1.00× (default)	215.95	—
	1.25×	209.90	−2.8%
	1.50×	199.97	−7.4%
	2.00×	174.06	−19.4%

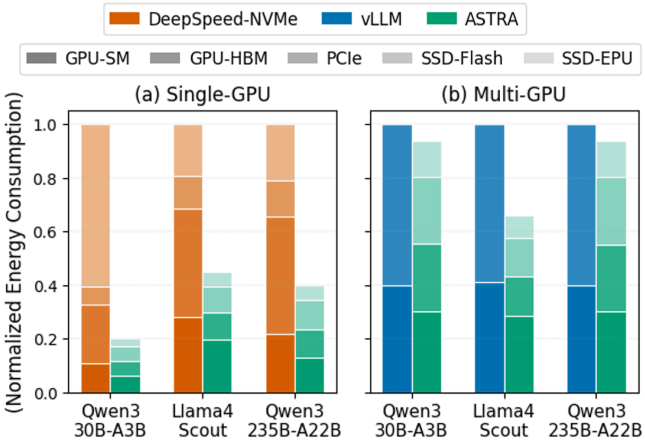


Figure 16: Normalized energy consumption breakdown for single- and multi-GPU MoE inference. (a) DeepSpeed-NVMe vs. ASTRA in the single-GPU setting, normalized to DeepSpeed-NVMe. (b) vLLM vs. ASTRA in the multi-GPU setting, normalized to vLLM. Bars are decomposed into GPU-SM, GPU-HBM, PCIe, SSD-Flash, and SSD-EPU energy.

lightweight activations. In the multi-GPU setting, ASTRA-MoE reduces per-token energy by up to 33.9% (LLaMa4-Scout at 128K; median 6.3% across evaluated models) over vLLM. Although SSD-side execution adds SSD-Flash and SSD-EPU energy, the reduction in GPU-SM and GPU-HBM energy dominates because ASTRA-MoE frees GPU HBM from resident expert weights and enables larger decode batches.

6 Related Works

MoE Inference Optimization. Prior MoE inference optimizations reduce model size, communication, or GPU memory pressure. Model-level methods compress or quantize MoE parameters, including EAQuant [11], MxMoE [9], MoE-I2 [51], and Slim-MoE [30]. System-level methods improve distributed MoE execution through expert parallelism, placement, prediction, and compute–communication overlap, including DeepSpeed-MoE [39], Occult [31],

Speculative MoE [29], and COMET [53]. Memory-centric systems such as SwapMoE [24], Expert Buffering [14], MoE-Infinity [50], FlashMoE [18], Diff-MoE [28], and FineMoE [52] exploit expert reuse through caching, prefetching, or swapping, ranking experts by access recency, frequency, or predicted reuse. However, these systems still execute selected experts on the GPU after moving their weights into HBM: non-resident experts are cached out, but migrate back to the GPU for execution. In contrast, ASTRA-MoE keeps heavily routed hot experts on the GPU but offloads memory-bound cold-expert computation itself to the ASTRA-SSD, reducing both GPU memory pressure and repeated weight movement.

Near-Data MoE Execution. MoNDE [22] targets MoE models whose expert parameters exceed GPU HBM capacity. Instead of repeatedly transferring offloaded expert weights from host memory to the GPU, MoNDE keeps hot experts on the GPU and executes cold experts on a CXL-attached, DRAM-based near-data processing device, exchanging activations rather than expert weights. HD-MoE [15] targets a memory-constrained 3D-stacked near-memory processing architecture and optimizes MoE execution through hybrid tensor-/expert-parallel mapping and online dynamic scheduling. Neither system targets NAND-flash-based SSDs. Compared with DRAM-based near-memory tiers, NAND flash provides substantially larger, lower-cost capacity, allowing ASTRA-MoE to keep the complete expert pool storage-resident. The distinction from ASTRA-MoE extends beyond the type of memory used. MoNDE uses near-data execution to mitigate the parameter-movement overhead that arises when the full expert pool cannot reside in GPU HBM, while HD-MoE optimizes computation mapping and scheduling under constrained near-memory capacity. In our multi-GPU setting, however, the expert pool can already fit in aggregate HBM. ASTRA-MoE proactively moves lightly routed expert computation to the SSD to reclaim GPU HBM for the KV cache and larger decode batches, while retaining heavily routed experts on the GPU to avoid overloading the constrained SSD-side execution path. Thus, the GPU–SSD partition in ASTRA-MoE is not simply a capacity-driven offloading decision; it jointly considers routed-token load, HBM available for the KV cache, attainable batch size, and the relative GPU/SSD execution cost.

In-/Near-Storage Processing for Efficient Deep Learning. In-/near-storage processing reduces data movement by placing computation closer to storage-resident data. Prior systems offload compressed training intermediates [3], place compute units near flash for large-model training [21], or accelerate memory-heavy inference operations such as attention and KV-cache handling [12, 16, 37]. AiF [27] offloads dense-model QKV and FFN computation to in-flash processing, and SmartRec [45] uses onboard FPGAs for recommendation inference. These works demonstrate the value of reducing data movement, but they do not address MoE-specific routing skew, where hot experts can overload a constrained storage-side accelerator. ASTRA-MoE is specialized for sparse MoE expert execution: it combines token-load-aware GPU–SSD placement with tile-level Flash–EPU pipelining, enabling a near-conventional SSD with an edge-class EPU to improve MoE inference efficiency under SSD power and area constraints. Unlike prior dense-model and attention-oriented in-/near-storage accelerators, ASTRA-MoE must account for input-dependent expert-load imbalance when deciding where computation should execute. Moreover, its storage-side execution is co-designed with NAND-specific behaviors through TEP, which overlaps page-granular flash accesses with EPU execution and tolerates variable flash completion latency.

7 Conclusion

Large-scale Mixture-of-Experts (MoE) models exceed GPU HBM limits, and existing GPU-centric solutions suffer from data-transfer overhead, limited batch-size scaling, and MoE Stall under skewed expert routing. ASTRA-MoE mitigates these bottlenecks by offloading memory-bound cold-expert computation to a specialized SSD while retaining heavily routed hot experts on the GPU. It leverages Temporal Hot-Expert Replacement (THER) for load-aware GPU expert residency and Tile-based Expert Processing (TEP) to overlap Flash reads with EPU computation and improve robustness to Flash-level latency variation. Evaluation confirms that ASTRA-MoE delivers efficient, high-throughput, and low-power inference for long-context MoE models.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (Nos. RS-2023-00260527 and RS-2025-00561961), and by Samsung Electronics Co., Ltd. (IO221213-04119-01). The EDA Tool was supported by the IC Design Education Center (IDEC), Korea.

References

[1] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S Gulavani, Ramachandran Ramjee, and Alexey Tumanov. 2024. Vidur: A large-scale simulation framework for llm inference. *Proceedings of Machine Learning and Systems* 6 (2024), 351–366.

[2] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, et al. 2022. Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[3] Jonghyun Bae, Jongsung Lee, Yunho Jin, Sam Son, Shine Kim, Hakbeom Jang, Tae Jun Ham, and Jae W Lee. 2021. FlashNeuron: SSD-Enabled Large-Batch training of very deep neural networks. In *19th USENIX Conference on File and Storage Technologies (FAST 21)*. 387–401.

[4] Weilin Cai, Juyong Jiang, Le Qin, Junwei Cui, Sunghun Kim, and Jiayi Huang. 2024. Shortcut-connected expert parallelism for accelerating mixture-of-experts. *arXiv preprint arXiv:2404.05019* (2024).

[5] William J Dally, Yatish Turakhia, and Song Han. 2020. Domain-specific hardware accelerators. *Commun. ACM* 63, 7 (2020), 48–57.

[6] Paul Delestrac, Jonathan Miquel, Debjyoti Bhattacharjee, Diksha Moolchandani, Francky Catthoor, Lionel Torres, and David Novo. 2024. Analyzing GPU Energy Consumption in Data Movement and Storage. In *2024 IEEE 35th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 143–151.

[7] Lishuo Deng, Shaojie Xu, Jinwu Chen, Changwei Yan, Jiajie Wang, Zhe Jiang, and Weiwei Shan. 2025. KVNAND: Efficient On-Device Large Language Model Inference Using DRAM-Free In-Flash Computing. *arXiv preprint arXiv:2512.03608* (2025).

[8] Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. 2022. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*. PMLR, 5547–5569.

[9] Haojie Duanmu, Xiuhong Li, Zhihang Yuan, Size Zheng, Jiangfei Duan, Xingcheng Zhang, and Dahua Lin. 2025. MxMoE: Mixed-precision Quantization for MoE with Accuracy and Performance Co-Design. *arXiv preprint arXiv:2505.05799* (2025).

[10] Artyom Eliseev and Denis Mazur. 2023. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238* (2023).

[11] Zhongqian Fu, Ning Ding, Kai Han, Xianzhi Yu, Xiaosong Li, Xinghao Chen, Yehui Tang, and Yunhe Wang. 2025. EAQuant: Enhancing Post-Training Quantization

for MoE Models via Expert-Aware Optimization. *arXiv preprint arXiv:2506.13329* (2025).

[12] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2024. Cost-Efficient large language model serving for multi-turn conversations with CachedAttention. In *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. 111–126.

[13] Google Coral. 2026. Coral Edge TPU Accelerator Module. https://www.coral.ai/products/accelerator-module. Accessed: 2026-04-28.

[14] Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Shruti Bhosale, Hsien-Hsin Lee, Carole-Jean Wu, and Benjamin Lee. 2024. Toward efficient inference for mixture of experts. *Advances in Neural Information Processing Systems* 37 (2024), 84033–84059.

[15] Haochen Huang, Shuzhang Zhong, Zhe Zhang, Shuangchen Li, Dimin Niu, Hongzhong Zheng, Runsheng Wang, and Meng Li. 2025. HD-MoE: Hybrid and Dynamic Parallelism for Mixture-of-Expert LLMs with 3D Near-Memory Processing. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[16] Hongsun Jang, Siung Noh, Changmin Shin, Jaewon Jung, Jaeyong Song, and Jinho Lee. 2025. INF^2: High-Throughput Generative Inference of Large Language Models using Near-Storage Processing. *arXiv preprint arXiv:2502.09921* (2025).

[17] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).

[18] Byeongju Kim, Jungwan Lee, Donghyeon Han, Hoi-Jun Yoo, and Sangyeob Kim. 2026. FlashMoE: Reducing SSD I/O Bottlenecks via ML-Based Cache Replacement for Mixture-of-Experts Inference on Edge Devices. *arXiv preprint arXiv:2601.17063* (2026).

[19] Byungryul Kim, Seungpil Lee, Beomseok Hah, Kangwoo Park, Yongsoon Park, Kangwook Jo, et al. 2023. A High-Performance 1Tb 3b/Cell 3D-NAND Flash with a 194MB/s Write Throughput on over 300 Layers. In *2023 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 27–29. doi:10.1109/ISSCC42615.2023.10067666

[20] Hyeunjoo Kim, Sanghun Oh, Jaeyong Lee, and Jihong Kim. 2025. Lazy Discharge: A High-Speed Energy-Efficient Read Technique for NAND Flash Memory. *IEEE Design & Test* (2025).

[21] 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.

[22] Taehyun Kim, Kwanseok Choi, Youngmock Cho, Jaehoon Cho, Hyuk-Jae Lee, and Jaewoong Sim. 2024. MoNDE: Mixture of Near-Data Experts for Large-Scale Sparse Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*. IEEE.

[23] Yechan Kim, Hwijoon Lim, and Dongsu Han. 2024. Scaling beyond the GPU memory limit for large mixture-of-experts model training. In *Forty-first International Conference on Machine Learning*.

[24] Rui Kong, Yuanchun Li, Qingtian Feng, Weijun Wang, Xiaozhou Ye, Ye Ouyang, Linghe Kong, and Yunxin Liu. 2023. SwapMoE: Serving Off-the-shelf MoE-based Large Language Models with Tunable Memory Budget. *arXiv preprint arXiv:2308.15030* (2023).

[25] Jaewook Kwak, Sangjin Lee, Kibin Park, Jinwoo Jeong, and Yong Ho Song. 2020. Cosmos+ openssd: Rapid prototype for flash storage systems. *ACM Transactions on Storage (TOS)* 16, 3 (2020), 1–35.

[26] Youngmin Kwon, Mingyeong Lee, Yifan Cai, Zhuohan Li, Tri Dao, Ion Stoica, and Matei Zhang. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. *arXiv preprint arXiv:2309.06180* (2023). https://arxiv.org/abs/2309.06180

[27] 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 Symposium on Computer Architecture*. 529–543.

[28] Kexin Li, Wenkan Huang, Qinggang Wang, Long Zheng, Xiaofei Liao, Hai Jin, and Jingling Xue. 2025. Diff-MoE: Efficient Batched MoE Inference with Priority-Driven Differential Expert Caching. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. doi:10.1145/3712285.3759903

[29] Yan Li, Pengfei Zheng, Shuang Chen, Zewei Xu, Yuanhao Lai, Yunfei Du, and Zhengang Wang. 2025. Speculative MoE: Communication Efficient Parallel MoE Inference with Speculative Token and Expert Pre-scheduling. *arXiv preprint arXiv:2503.04398* (2025).

[30] Zichong Li, Chen Liang, Zixuan Zhang, Ilgee Hong, Young Jin Kim, Weizhu Chen, and Tuo Zhao. 2025. SlimMoE: Structured Compression of Large MoE Models via Expert Slimming and Distillation. *arXiv preprint arXiv:2506.18349* (2025).

[31] Shuqing Luo, Pingzhi Li, Jie Peng, Hanrui Wang, Yu Cheng, Tianlong Chen, et al. 2025. Occult: Optimizing Collaborative Communication across Experts for Accelerated Parallel MoE Training and Inference. *arXiv preprint arXiv:2505.13345* (2025).

[32] NVIDIA. 2018. *NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics*. https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/ Accessed: 2026-02-07.

[33] NVIDIA Corporation. 2025. NVIDIA Nsight Systems. https://developer.nvidia.com/nsight-systems. Accessed: 2025-10-09.

[34] NVIDIA Corporation. 2026. *GPUDirect RDMA*. NVIDIA Corporation. https://docs.nvidia.com/cuda/gpudirect-rdma/ Version 13.2.

[35] NVIDIA Corporation. 2026. *GPUDirect Storage*. NVIDIA Corporation. https://docs.nvidia.com/gpudirect-storage/

[36] ONFI Working Group. 2026. Open NAND Flash Interface Specification, Revision 6.0. https://onfi.org/specs.html.

[37] Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. 2025. InstAttention: In-Storage Attention Offloading for Cost-Effective Long-Context LLM Inference. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1510–1525.

[38] Bharadwaj Pudipeddi, Maral Mesmakhosroshahi, Jinwen Xi, and Sujeeth Bharadwaj. 2020. Training large neural networks with constant memory using a new execution algorithm. *arXiv preprint arXiv:2002.05645* (2020).

[39] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*. PMLR, 18332–18346.

[40] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. Zero-offload: Democratizing billion-scale model training. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. 551–564.

[41] Ananda Samajdar, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. 2018. Scale-sim: Systolic cnn accelerator simulator. *arXiv preprint arXiv:1811.02883* (2018).

[42] Samsung Semiconductor. 2024. Samsung LPDDR5X Chip Specification. https://semiconductor.samsung.com/dram/lpddr/lpddr5x/. Accessed: 2025-07-02.

[43] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538* (2017).

[44] Liang Shen, Zhihua Wu, WeiBao Gong, Hongxiang Hao, Yangfan Bai, HuaChao Wu, Xinxuan Wu, Jiang Bian, Haoyi Xiong, Dianhai Yu, et al. 2022. Se-moe: A scalable and efficient mixture-of-experts distributed training and inference system. *arXiv e-prints* (2022), arXiv–2205.

[45] Mohammadreza Soltaniyeh, Veronica Lagrange Moutinho Dos Reis, Matt Bryson, Xuebin Yao, Richard P Martin, and Santosh Nagarakatte. 2022. Near-storage processing for solid state drive based recommendation inference with smartssds®. In *Proceedings of the 2022 ACM/SPEC on International Conference on Performance Engineering*. 177–186.

[46] Weiyi Sun, Mingyu Gao, Zhaoshi Li, Aoyang Zhang, Iris Ying Chou, Jianfeng Zhu, Shaojun Wei, and Leibo Liu. 2025. Lincoln: Real-Time 50~100B LLM Inference on Consumer Devices with LPDDR-Interfaced, Compute-Enabled Flash Memory. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1734–1750.

[47] Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. 2024. Hobbit: A mixed precision expert offloading system for fast moe inference. *arXiv preprint arXiv:2411.01433* (2024).

[48] Arash Tavakkol, Juan Gómez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu. 2018. MQSim: A framework for enabling realistic studies of modern Multi-QueueSSD devices. In *16th USENIX Conference on File and Storage Technologies (FAST 18)*. 49–66.

[49] Yvan Tortorella, Luca Bertaccini, Davide Rossi, Luca Benini, and Francesco Conti. 2022. RedMulE: A compact FP16 matrix-multiplication accelerator for adaptive deep learning on RISC-V-based ultra-low-power SoCs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1099–1102.

[50] Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. 2025. MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache. https://openreview.net/forum?id=BL7WMLJKZM

[51] Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. 2024. MoE-I^2: Compressing Mixture of Experts Models through Inter-Expert Pruning and Intra-Expert Low-Rank Decomposition. *arXiv preprint arXiv:2411.01016* (2024).

[52] Hanfei Yu, Xingqi Cui, Hong Zhang, Hao Wang, and Hao Wang. 2026. Taming Latency-Memory Trade-Off in MoE-Based LLM Serving via Fine-Grained Expert Offloading. In *Proceedings of the 21st European Conference on Computer Systems (EuroSys)*. doi:10.1145/3767295.3769319

[53] Shulai Zhang, Ningxin Zheng, Haibin Lin, Ziheng Jiang, Wenlei Bao, Chengquan Jiang, Qi Hou, Weihao Cui, Size Zheng, Li-Wen Chang, et al. 2025. Comet: Fine-grained computation-communication overlapping for mixture-of-experts. *arXiv preprint arXiv:2502.19811* (2025).

[54] Zheng Zhang, Yaqi Xia, Hulin Wang, Donglin Yang, Chuang Hu, Xiaobo Zhou, and Dazhao Cheng. 2024. Mpmoe: Memory efficient moe for pre-trained models

with adaptive pipeline parallelism. *IEEE Transactions on Parallel and Distributed Systems* (2024).

[55] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. 2022. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems* 35 (2022), 7103–7114.

[56] Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. 2022. St-moe: Designing stable and transferable sparse expert models. *arXiv preprint arXiv:2202.08906* (2022).