BEYOND OVERLAP: OS-AWARE, BANDWIDTH-CENTRIC EXPERT STREAMING FOR MOE INFERENCE ON UNIFIED-MEMORY SYSTEMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) models reduce per-token computation but can exceed the memory capacity of unified-memory devices, forcing router-selected experts to be streamed from SSD. We show that the resulting bottleneck is not expert computation or unified-memory bandwidth, but the realized bandwidth of the OS-mediated SSD-to-unified-memory path. XSTREAM increases this bandwidth by issuing routed-expert requests concurrently, jointly tuning request granularity and I/O/decompression parallelism, and streaming each expert through lossless decompression, the Gate and Up projections, and the dependent Down projection without waiting for the whole expert. On an NVIDIA GB10 system, it achieves 2.07–$4.65\times$ higher throughput than ZipMoE at matched memory budgets, and 1.47–$1.53\times$ higher throughput than serial one-expert-ahead execution without pipelining. These results demonstrate that, when experts must be fetched, exposing and consuming storage parallelism is a first-order requirement for SSD-backed MoE inference.

1 INTRODUCTION

MoE models activate only a router-selected top-K subset of their experts per token, which reduces compute but not total parameter storage (Fedus et al., 2022; Jiang et al., 2024; Dai et al., 2024). On a unified-memory device whose inference budget cannot hold all experts, a cache miss places the routed path on SSD:

$$\text{NVMe SSD} \rightarrow \text{OS I/O stack} \rightarrow \text{unified memory} \rightarrow \text{MoE computation.}$$

For Qwen1.5, MoE work occupies 93.3% of layer time yet uses only 18.4 GB/s, or 8.0% of a 230 GB/s memory reference; GB10's SSD reaches 6.85 GB/s in a pure-read test (Table 11). In a separate no-cache serial profile, useful resident-equivalent expert compute accounts for only 10.6% of MoE elapsed time; roughly 89% lies outside that computation, primarily in the storage-bound path (Appendix A). This is a time breakdown, not a hardware GPU-utilization measurement.

Prior frameworks reduce expert movement through caching, prediction, compression, or prefetching (Sheng et al., 2023; Xue et al., 2024; He et al., 2026; Yang et al., 2026). These methods are complementary to our focus: when a router-selected expert is not resident, the runtime must still make its SSD payload available quickly. A conventional one-expert-ahead overlap exposes only a small amount of independent I/O and can leave an NVMe device idle even while its computation graph appears overlapped.

XSTREAM submits all K routed-expert reads concurrently, calibrates chunk size and I/O concurrency for a model/platform/workload profile, and streams decompressed blocks into the Gate/Up–SwiGLU–Down dataflow without a whole-expert barrier. Figure 1 gives the overall design. It keeps the router-selected experts and their BF16 weights unchanged; the schedule changes when data is requested, decoded, and consumed. A bitwise comparison of our schedules is reported in Appendix A.

Our contributions are: (i) an OS-aware storage-path design that exposes the router-selected miss set as concurrent SSD work; (ii) one-time joint calibration of chunk size and I/O concurrency; and (iii) an operator-aware streaming pipeline that overlaps reads, lossless GPU decompression, Gate/Up–SwiGLU, and blockwise Down products.

On GB10, XSTREAM achieves 2.07–$4.65\times$ the matched-memory throughput of ZipMoE and 1.47–$1.53\times$ the throughput of serial one-expert-ahead fetching without pipelining. On Radeon/Windows

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

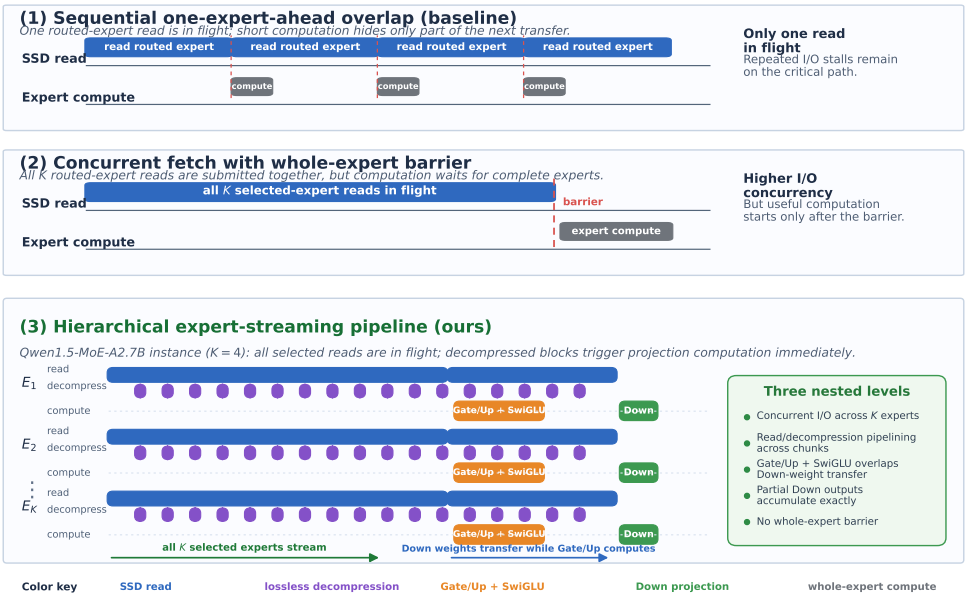

Figure 1: XSTREAM overview. Our operator-aware pipeline keeps all selected-expert reads in flight, decompresses blocks on arrival, and starts Gate/Up–SwiGLU and blockwise Down computation without a whole-expert barrier.

Table 1: Published MoE offloading settings and their primary mechanisms.

System	Weight source	Memory topology	Primary mechanism
MoE-Infinity	Host DRAM	Discrete DRAM/VRAM	Prediction and caching
Fiddler	Host DRAM	Discrete DRAM/VRAM	CPU/GPU expert execution
Klotski	DRAM + disk	Discrete DRAM/VRAM	Multi-batch prefetch
ZipMoE	NVMe SSD	Unified CPU/GPU RAM	Compression and caching
XSTREAM	NVMe SSD	Unified CPU/GPU RAM	Concurrent reads and GPU streaming

with an 8 GB all-miss cap, it reduces end-to-end time between tokens by 1.69–1.95× and 256-token time to first token by 1.72–2.13× relative to raw serial execution.

2 RELATED WORK

In-memory MoE systems such as Tutel and MegaBlocks optimize routed GPU execution after weights are resident (Hwang et al., 2022; Gale et al., 2023). Memory-constrained systems instead reduce expert movement through adaptive routing, caching, prediction, or prefetching (Zhong et al., 2024; Skliar et al., 2025; Xue et al., 2024; He et al., 2026). These approaches are complementary: XSTREAM preserves the original top-K routing and optimizes the unavoidable cache misses.

Prior pipelines also overlap movement and computation (Kamahori et al., 2025; Fang et al., 2025), but short MoE computation cannot hide SSD service when only one next-expert read is active. ZipMoE is the closest system: it combines cache-centric execution with lossless bit-field compression, one dedicated SSD I/O thread, CPU decompression workers, and GPU recovery (Yang et al., 2026). In contrast, XSTREAM submits the selected miss set concurrently, decodes ANS chunks on the GPU, and consumes ready Gate/Up and Down blocks before an expert is complete. Thus we maximize the realized bandwidth of SSD I/O rather than only reducing its volume or hiding CPU decode.

Table 1 summarizes published evaluations (Xue et al., 2024; Kamahori et al., 2025; Fang et al., 2025; Yang et al., 2026) rather than all capabilities of later code: MoE-Infinity's current repository also supports SSD offload and states that its implementation differs from the paper (MoE-Infinity Contributors, 2026). Among the systems in Table 1, ZipMoE is the closest published baseline evaluated with SSD offload on a unified-memory device. We therefore compare against ZipMoE under matched memory budgets; MoE-Infinity, Fiddler, and Klotski report results for separate CPU and GPU memory.

Open-source engines also stream experts from storage. Colibrì (Fornaro & Colibrì contributors, 2026) places experts across VRAM, RAM, and NVMe, uses a bounded asynchronous I/O pool with one-layer-ahead prefetch, and includes an experimental GPU ANS tier. Its documented load path reads each expert's adjacent weight matrices together in one I/O request. XSTREAM instead consumes decoded Gate/Up and Down blocks as they become ready, allowing computation before the whole expert has arrived.

3 BACKGROUND: MOE LAYER AND EXPERT DEPENDENCIES

For a token hidden state $\mathbf{x} \in \mathbb{R}^d$, the router produces a score for every routed expert and selects the top-K set $\mathcal{S}(\mathbf{x})$. The selected scores are normalized to routing weights p_i, so only the experts in $\mathcal{S}(\mathbf{x})$ are evaluated. The routed path is therefore

$$\mathbf{r}(\mathbf{x}) = \sum_{i \in \mathcal{S}(\mathbf{x})} p_i E_i(\mathbf{x}). \tag{1}$$

Many modern MoE architectures additionally include a *shared expert* E_s, which is evaluated for every token rather than selected by the router, to capture common knowledge separately from the routed experts (Dai et al., 2024). The layer output is

$$\mathbf{y}(\mathbf{x}) = \mathbf{r}(\mathbf{x}) + E_s(\mathbf{x}). \tag{2}$$

Thus, sparse routing reduces the number of routed experts evaluated per token, but it does not remove the need to access the parameters of each selected expert. The shared expert follows a fixed, always-active path; our focus is the routing-dependent path through the selected experts, whose parameter availability varies from token to token.

Each selected expert is a gated feed-forward network. For expert i, its Gate and Up projections can be written as

$$\mathbf{g}_i = \mathbf{x}W_i^g, \qquad \mathbf{u}_i = \mathbf{x}W_i^u, \tag{3}$$

followed by the SwiGLU activation and the Down projection,

$$E_i(\mathbf{x}) = (\mathrm{SiLU}(\mathbf{g}_i) \odot \mathbf{u}_i)\, W_i^d. \tag{4}$$

SwiGLU is a gated linear-unit variant used in Transformer feed-forward layers (Shazeer, 2020). Here W_i^g, W_i^u, and W_i^d are the Gate, Up, and Down weights of expert i, respectively. The Gate and Up projections are independent once their weights are available, whereas the elementwise SwiGLU operation depends on both outputs; the Down projection in turn depends on the resulting intermediate activation. These dependencies define when partial expert data can be consumed and will guide our operator-aware pipeline.

4 METHOD

4.1 CONCURRENT ROUTED-EXPERT I/O

At each MoE layer, the router identifies K routed experts whose weights must be available to evaluate Eq. 1. When these experts are not resident in unified memory, the model execution engine must retrieve their compressed weights from SSD. Let $t_{\mathrm{transfer},i}$ denote the time from submitting the read for expert i until its data is available for computation, and let $t_{\mathrm{compute},i}$ denote the corresponding expert-computation time. In the latency-oriented decode setting considered here, we observe

$$t_{\mathrm{transfer},i} > t_{\mathrm{compute},i}, \qquad \forall i \in \mathcal{S}(\mathbf{x}). \tag{5}$$

A natural baseline reads one selected expert at a time and overlaps the computation of expert i with the read of expert $i+1$. This overlap is insufficient under Eq. 5: after the computation of i completes, a remaining portion of the read for $i+1$ is still on the critical path. The same bubble recurs for every selected expert, as shown in Fig. 1(1). Thus, sequential execution does not convert an I/O-dominated layer into a compute-hidden one; it merely hides a short portion of each next-expert transfer.

Our first mechanism therefore submits the independent reads for all K selected experts concurrently. Let $T_{\mathrm{seq}}^{\mathrm{read}}$ and $T_{\mathrm{conc}}^{\mathrm{read}}$ denote the wall-clock time to make the same selected expert payloads available under sequential and concurrent reads, respectively. We define the transfer time saved by concurrent submission as

$$\Delta t_{\mathrm{transfer}} \equiv T_{\mathrm{seq}}^{\mathrm{read}} - T_{\mathrm{conc}}^{\mathrm{read}}. \tag{6}$$

More generally, the concurrent-I/O regime of interest is characterized by

$$\Delta t_{\text{transfer}} > K \max_{i \in \mathcal{S}(\mathbf{x})} t_{\text{compute},i}. \tag{7}$$

That is, the read-time reduction alone is larger than the total amount of selected-expert computation that a sequential one-expert-ahead schedule could attempt to hide. Qwen1.5-MoE-A2.7B is one instantiation: it has $K = 4$ router-selected experts, and we measure $\Delta t_{\text{transfer}} > 4t_{\text{compute},i}$ for every selected expert, with a 2.56 ms saving for each MoE-layer (MoE-block) invocation relative to the one-expert-ahead baseline.

To expose independent storage requests, we use a pool of host I/O workers on both Linux and Windows. A worker submits reads for expert chunks; N is the number of workers, not a promised device queue depth. In an all-miss Qwen1.5 end-to-end GB10 run, eight workers sustain mean device queue size `aqu-sz`=46.5, 4.87 GB/s application-read throughput, and 68.7% device utilization. This end-to-end value includes pipeline activity and is distinct from the payload-stage bandwidth sweep in Appendix C. The Linux and Windows read APIs and the distinction between I/O workers and NVMe queueing are detailed in Appendix A.

4.2 Lossless ANS Compression

We store nonresident weights with asymmetric numeral systems (ANS), a lossless entropy coder (Duda, 2013). It reduces storage traffic whenever the weight-symbol distribution is nonuniform, while reconstructing the original BF16 values exactly.

Compression is beneficial only when the traffic saved on the slow SSD-to-RAM path exceeds its decompression and scheduling costs. Let W_M be the total uncompressed volume of the K_M selected experts for model M, $\rho_M(C)$ the measured ANS compression ratio, $B_{\text{SSD}\to\text{RAM}}$ the realized storage-path bandwidth, and B_{RAM} the effective RAM/unified-memory throughput. A first-order model of transfer and decoding costs is

$$\begin{aligned} T_{\text{raw}} &\approx W_M \left(\frac{1}{B_{\text{SSD}\to\text{RAM}}(N; \mathcal{O}, \text{SSD})} + \frac{1}{B_{\text{RAM}}} \right), \\ T_{\text{ANS}}(C, N) &\approx \frac{W_M}{\rho_M(C)} \left(\frac{1}{B_{\text{SSD}\to\text{RAM}}(N; \mathcal{O}, \text{SSD})} + \frac{1}{B_{\text{RAM}}} \right) + D_{\text{ANS}}(C) + H(C, N), \end{aligned} \tag{8}$$

where D_{ANS} is lossless-decompression time and H captures chunk bookkeeping, kernel launches, and queueing. These terms describe isolated costs, not the exact latency of concurrent execution on unified memory: I/O and GPU decode may interact through shared memory resources. Because SSD-to-RAM bandwidth is much lower than RAM/unified-memory throughput in our target system, the first term is the dominant opportunity for compression; whether ANS wins in the running pipeline is determined by measured latency. On Qwen1.5-MoE-A2.7B, matched ANS loading is $1.229\times$ faster than raw BF16 with one I/O worker and $1.238\times$ faster with deeper I/O concurrency. The detailed codec sweep is in the appendix.

ANS decompression is GPU-resident: ready outer chunks are independent elements of a GPU decode batch. GB10 uses nvCOMP; Radeon uses our native GPU decoder for the same lossless format (NVIDIA Corporation, 2025). The outer chunk C is therefore a measured streaming/decode trade-off, not a serial per-chunk cost. For a compressed deployment, C is chosen offline and fixed during inference. Changing it requires re-encoding the weights at the new chunk size and replacing the SSD files and decode metadata; we do not store every candidate encoding for serving.

4.3 Platform-Specific Chunk and Worker Calibration

Equation 8 shows why a universal (C, N) is not well-defined. W_M and K_M depend on the model and routing configuration; $\rho_M(C)$ depends on weight-symbol statistics; and the realized storage bandwidth depends on the SSD, operating-system/NVMe path, request granularity, and offered concurrency. We therefore select an operating point from measured MoE-stage latency for a specified deployment profile—model, storage path, workload, and cache budget—rather than tune chunk size and worker count independently:

$$(C^*, N^*) = \arg\min_{C,N} T_{\text{MoE}}(C, N), \tag{9}$$

where T_{MoE} includes SSD reads, lossless decompression, weight reconstruction, and expert computation for one MoE layer; it is not full-token latency. Equation 8 motivates the cost terms, while

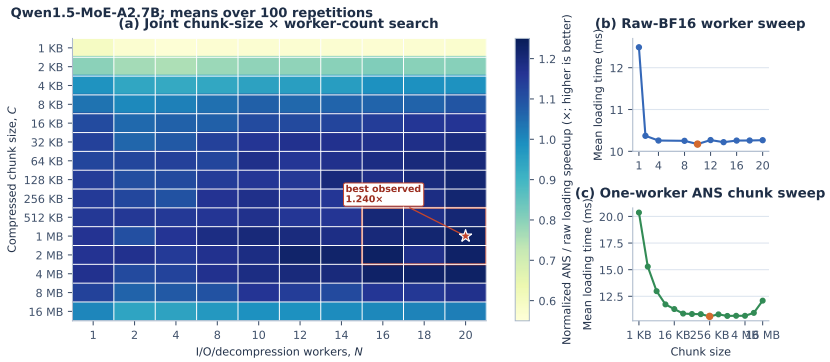

Figure 2: Qwen1.5-MoE-A2.7B all-miss loading sweep (100 repetitions). (a) ANS/raw speedup at matched worker counts; (b) raw worker sweep; (c) one-worker ANS chunk sweep. Box: high-performance region; star: minimum absolute loading latency.

Eq. 9 uses measured joint execution. We target batch-one inference for the model's K routed experts when calibrating the basic miss path; cached end-to-end workloads are checked separately. The selected pair is then reused for subsequent requests with that profile. An order-randomized GB10 timing check at $C = 1$ MB and $N = 8$ detected no statistically significant decode slowdown during concurrent reads in that setting (Appendix C.4); it does not establish that shared-memory interference is absent elsewhere.

Figure 2 shows an all-miss, single-layer Qwen1.5-MoE-A2.7B *loading-stage* diagnostic, with $T_{\text{load}}(C, N)$ measuring SSD reads plus ANS decode but excluding expert computation. Each cell is the mean of 100 repetitions and is normalized to raw loading at the same worker count. The star is also the minimum measured absolute loading latency: 8.274 ms at $C = 1$ MB and $N = 20$. Independently assembling the best chunk-only and worker-only selections reaches 8.635 ms. Joint calibration therefore removes 0.361 ms (4.18%) from this isolated selected-expert loading experiment. The heatmap's broad high-performance region reinforces that the calibration is a practical one-time selection, rather than a claim that worker count improves latency monotonically. In a separate Qwen1.5 sensitivity test, using chunk size and worker count optimized for fetching four experts to fetch eight experts is within 0.15% of the setting optimized for eight experts on that same eight-expert workload (Appendix C.4).

We record the complete-layer search as

$$\mathcal{L}_{M,\text{platform}} = \{(C, N, \overline{T}_{\text{MoE}}(C, N))\}, \qquad (C^*, N^*) = \arg \min_{(C,N)\in\mathcal{L}_{M,\text{platform}}} \overline{T}_{\text{MoE}}(C, N). \tag{10}$$

The Figure 2 loading sweep completes in about three minutes and is excluded from per-token inference latency; its recorded cells are T_{load}, not the T_{MoE} entries in Eq. 10. Cache hits change the number of outstanding read tasks, so the worker count selected for a cached end-to-end workload need not equal the all-miss single-layer diagnostic point in Fig. 2. Full-model schedule ablations therefore use the best end-to-end operating point selected once for their model, runtime, and cache budget.

4.4 Hierarchical Expert-Streaming Pipeline

Concurrent routed-expert I/O removes the shallow-queue limitation, but it does not by itself remove a *whole-expert barrier*: a runtime may still wait until every weight tensor of an expert has arrived and been decompressed before beginning its feed-forward computation. We remove this second barrier with a *hierarchical expert-streaming pipeline*. Figure 1 contrasts three schedules: (1) the sequential one-expert-ahead baseline; (2) concurrent fetch with a whole-expert barrier; and (3) our pipeline, which combines inter-expert concurrency with chunk- and projection-level dataflow.

For an exact blockwise formulation, partition the intermediate dimension of expert i into disjoint blocks $\{\mathcal{I}_b\}_{b=1}^{B_i}$. The corresponding Gate, Up, and Down weights are

$$W_{i,b}^g = W_i^g[:, \mathcal{I}_b], \quad W_{i,b}^u = W_i^u[:, \mathcal{I}_b], \quad W_{i,b}^d = W_i^d[\mathcal{I}_b, :]. \tag{11}$$

Once the losslessly decompressed Gate/Up block b is available, the runtime can compute its SwiGLU activation without any other block:

$$\mathbf{z}_{i,b} = \mathrm{SiLU}(\mathbf{x}W^g_{i,b}) \odot (\mathbf{x}W^u_{i,b}), \qquad E_i(\mathbf{x}) = \sum_{b=1}^{B_i} \mathbf{z}_{i,b}W^d_{i,b}. \tag{12}$$

Equation 12 is algebraically identical to Eq. 4: the blocks partition only the intermediate dimension and their Down-projection contributions are summed. In finite precision, however, a different accumulation order can change low-order bits. The implementation orders completed expert results by expert ID before the final `index_add_` scatter, rather than accumulating them in I/O completion order. This preserves a deterministic expert-level reduction order while allowing reads and expert work to finish out of order. The scope of our bitwise schedule comparison is given in Appendix A.

The runtime prioritizes Gate/Up blocks so that a first block can produce $\mathbf{z}_{i,b}$ early. It then continues to stream and decompress Down blocks while Gate/Up and SwiGLU computation proceeds. Let $\tau^z_{i,b}$ be the time at which $\mathbf{z}_{i,b}$ is available and $\tau^d_{i,b}$ the time at which the matching Down block is decompressed. The partial output becomes ready at

$$\begin{aligned} \mathbf{p}_{i,b} &= \mathbf{z}_{i,b}W^d_{i,b}, \\ \tau^p_{i,b} &= \max\{\tau^z_{i,b}, \tau^d_{i,b}\} + t^{\mathrm{down}}_{i,b}, \qquad E_i(\mathbf{x}) = \sum_b \mathbf{p}_{i,b}. \end{aligned} \tag{13}$$

The maximum in Eq. 13, rather than a wait for all B_i blocks, is the key dependency: Down computation for block b starts as soon as *its* activation and Down weights are ready. In parallel, later SSD reads and decompression continue. After the initial fill, the pipeline's throughput is therefore governed by the slowest active stage—I/O, decompression, Gate/Up–SwiGLU, or Down—rather than their sum for every block.

The hierarchy has three levels. First, all K routed experts have reads in flight concurrently, exposing I/O concurrency to the OS/NVMe path. Second, within each expert, decompressing block b overlaps the read of later blocks. Third, projection-aware dataflow overlaps continuing Down streaming with the Gate/Up–SwiGLU work enabled by earlier blocks. The implementation preserves out-of-order expert completion and has no whole-expert materialization barrier. The resulting latency reduction equals the work removed from the critical path by these overlaps; it is workload-dependent and is therefore measured end-to-end rather than inferred from the schematic. In the Qwen1.5-MoE-A2.7B instance, $K = 4$, but the schedule and equations remain parameterized by K and B_i.

5 Evaluation

5.1 Experimental Setup

We evaluate NVIDIA GB10 (20-core Arm CPU, Blackwell GPU, 119 GiB unified LPDDR5X) on Ubuntu 24.04.3 LTS and AMD Ryzen AI Max+ 395 (Radeon 8060S, 64 GiB unified LPDDR5X) on Windows 11 Pro 25H2. Both use a PCIe 4.0 $\times 4$ NVMe SSD path with a one-direction link peak of 7.88 GB/s; the GB10 pure-read microbenchmark reaches 6.85 GB/s (87% of that peak). Unless stated otherwise, weights use GPU-resident ANS lossless compression; GB10 uses nvCOMP and Radeon uses our native GPU decoder. We greedily decode 64 tokens. Schedule ablations use distinct ShareGPT conversations with approximately 256-token prompts and report the mean of three runs. The GB10 evaluation uses 8–16 GB runtime memory budgets; the Radeon validation uses an 8 GB all-miss cap. Further platform details are in Appendix A.

Our cache is a per-layer fixed-slot LRU cache of *decoded routed experts*. Each miss is admitted unconditionally; the least-recently-used slot that is not pinned by the current routed-expert group is evicted. Thus, a slot always holds one complete decoded expert and cannot be partially resident. The shared expert is part of the dense GPU-resident parameter set, is neither cached nor streamed, and is excluded from the reported routed-expert hit rates. A hit-rate lookup corresponds to one unique routed expert in one MoE layer during the measured generation window. For Qwen1.5, the 8, 12, and 16 GB cache regimes correspond to 11, 22, and 32 decoded-expert slots per layer, respectively; the resident control holds all 60 routed experts.

For each model and runtime, one-time calibration selects the end-to-end chunk and I/O-worker operating point; it is then held fixed across the corresponding schedule-ablation arms. A read segment

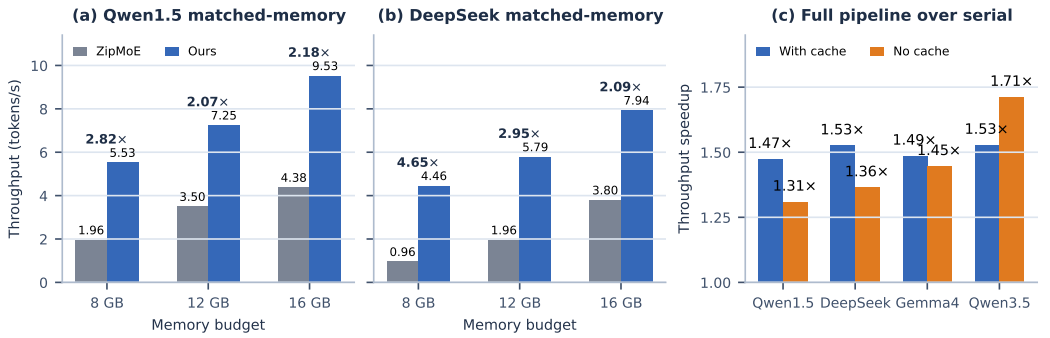

Figure 3: GB10 results. (a,b) Matched-memory throughput; labels show ours/ZipMoE speedup. (c) Full-pipeline speedup over serial fetching at 12 GB.

and an ANS outer chunk are distinct: the segment is independently encoded, while the outer chunk controls ANS decoding and streaming granularity.

Exact storage timings are inherently sensitive to OS wake-ups, background activity, and transient device queue state. We therefore control prompt, cache, and page-cache conditions within each comparison, report repeated-run means (and paired intervals where available), and interpret final decimal places as platform- and run-specific rather than portable constants. The calibration surface uses 100 repetitions; the GB10 schedule ablation uses a fixed protocol and three repetitions per cell.

We evaluate Qwen1.5-MoE-A2.7B, DeepSeek-V2-Lite, Gemma4-26B-A4B, and Qwen3.5-35B-A3B. They span top-$K = 4$–8, 60–256 routed experts, and 50.3–103.8 MB of selected BF16 weights per MoE layer; full model details are in Appendix A. We report within-platform improvements rather than absolute rates across devices.

For all SSD-backed runs, a selected expert follows SSD $\rightarrow$ a GPU-consumable unified-memory allocation $\rightarrow$ GPU computation; there is no separate physical VRAM. We avoid CPU entropy decoding and CPU-produced intermediate buffers. ZipMoE instead performs CPU exponent decode followed by GPU recovery; despite UMA zero-copy, this staging consumes shared-memory traffic and adds a dependency before the GEMM. All reported ZipMoE times include that cost (Yang et al., 2026).

We use two complementary comparisons. First, the schedule ablation holds the model, ShareGPT prompts truncated to 256 input tokens, cache regime, and decoding configuration fixed while changing only the execution schedule. Its memory budget is enforced by slot count, not by a cgroup cap. Second, the ZipMoE comparison uses a short fixed prompt and caps both systems with the same cgroup `MemoryMax` budget (with `MemorySwapMax=0`); the page cache is evicted before each launch. The matched-memory throughput figure gives ZipMoE the better of its two documented prefetcher settings ($\text{topk} \in \{0, 4\}$) at each budget. The detailed fixed-baseline protocol and storage profile are in Appendix A. These experiments answer different questions, so their absolute token rates are not compared directly. The cgroup cap prevents ZipMoE's offload file from remaining in the OS page cache, so every matched-memory result entails SSD reads. We exclude Qwen at 32 GB because our cache contains all experts whereas ZipMoE still reads from SSD; it is not a like-for-like storage-path setting.

5.2 End-to-End Results

Figure 3 summarizes GB10 results. Panels (a,b) directly compare the matched-memory throughput of ZipMoE and XSTREAM for Qwen1.5 and DeepSeek. ZipMoE is cache-centered: when the budget drops from 16 GB to 12 GB, its throughput falls by 20% on Qwen1.5 and 48% on DeepSeek; the corresponding reductions for our pipeline are 24% and 27%. Appendix D details this cache-miss behavior and the data-path and cache-budget distinctions. At matched Qwen1.5 budgets, XSTREAM reads 1.4–3.7$\times$ more SSD data yet realizes 4.0–8.4$\times$ ZipMoE's whole-run effective SSD bandwidth (Appendix A).

Figure 3(c) and Table 2 isolate the scheduling gain. The serial arm parallelizes chunks within one expert, but routed experts remain in serial order. Concurrent fetch offers the entire routed group to the I/O path but retains a whole-expert compute barrier; the full pipeline removes that barrier by starting Gate/Up–SwiGLU on ready blocks while Down weights stream. The Appendix B four-expert

Table 2: GB10 throughput (tokens/s; 12 GB budget).

Model	Serial no pipeline	ANS serial (one expert ahead)	Concurrent (no pipeline)	Full operator-aware	Full / serial no pipeline
Qwen1.5-MoE-A2.7B	4.026	4.721	4.782	5.930	1.473×
DeepSeek-V2-Lite	3.316	3.916	4.021	5.060	1.526×
Gemma4-26B-A4B	2.535	2.733	3.303	3.768	1.486×
Qwen3.5-35B-A3B	2.475	2.496	3.461	3.782	1.528×

Table 3: GB10 latency (s; 12 GB budget).

(a) Time to first token (TTFT, s)

Model	Serial no pipe.	ANS serial	Conc. no pipe.	Full
Qwen1.5	4.141	3.509	2.944	2.300
DeepSeek	5.342	4.157	3.755	2.758
Gemma4	5.752	4.939	3.673	3.124
Qwen3.5	9.755	9.660	5.918	5.112

(b) Time between tokens (TBT, s)

Model	Serial no pipe.	ANS serial	Conc. no pipe.	Full
Qwen1.5	0.1867	0.1595	0.1659	0.1348
DeepSeek	0.2221	0.1934	0.1932	0.1570
Gemma4	0.3097	0.2933	0.2493	0.2200
Qwen3.5	0.2556	0.2537	0.1996	0.1875

ablation confirms that grouping experts improves payload rate even after intra-expert threading. Concurrent fetch alone adds 1.3–2.7% for $K = 4$–6 and 20.9–38.7% for $K = 8$; full versus serial no-pipeline throughput is 1.47–$1.53\times$. The full design transfers to the second platform, reducing TTFT by 1.72–$2.13\times$ under an all-miss 8 GB cap (Appendix A).

"ANS serial" in Table 2 means the one-expert-ahead schedule *with* lossless ANS compression and GPU decoding; it is not a raw-BF16 baseline (Duda, 2013; NVIDIA Corporation, 2025). Table 2 uses the calibrated operating point selected for this end-to-end workload. The appendix ladders use a separate fixed configuration to decompose implementation steps; they are diagnostic runs, not numerical replicates of this table. The raw/no-compression serial path, exact worker/chunk settings, and the full stepwise decomposition are reported in Appendix B.

Tables 2 and 3 also show the 12 GB serial no-pipeline schedule. The Full/serial column divides the reported full rate by that serial rate for each model. The cold-miss and 8 GB serial measurements, including uncertainty and cache-hit rates, appear in Appendix E. These main-table schedule estimates use $n = 3$ runs per cell. Separate $n = 20$ rerun intervals cover the Full/serial gain for all four models in Appendix B (Table 14). The same table gives per-arm means and standard deviations for Qwen1.5 and DeepSeek.

5.3 Cross-Platform End-to-End Validation

We next test whether the design transfers beyond GB10/Linux using the same real model checkpoints on a Radeon 8060S (gfx1151) APU under Windows. These runs enforce an 8 GB runtime cap and use an all-miss policy: a selected expert is discarded after it is consumed and replaced by the next layer's selected expert, so no routed-expert reuse can serve a later layer or decoding step. At a 256-token context over 100 repetitions, the full ANS operator-aware pipeline improves end-to-end TBT by $1.69\times$–$1.95\times$ relative to raw serial execution; both compression and scheduling contribute, while the full configuration is the reported system result.

For TTFT, raw serial and ANS full-pipeline trials are paired, interleaved, and use the same deterministic expert-selection sequence for each repetition. At the saturated 256-token prefill point, we use 100 repetitions and report means, p99, and bootstrap 95% CIs for the paired speedup. We use 20 repetitions at 16 and 1,024 prompt tokens to bracket this point; detailed times and intervals are in Appendix A. We omit p99 at $n = 20$ because it is only the maximum. The TTFT reduction remains $1.72\times$–$2.13\times$ across the three prompt lengths. At 256 tokens, holding ANS fixed, the full operator-aware pipeline contributes a further $1.426\times$–$1.747\times$ reduction over ANS serial execution, exceeding the $1.206\times$–$1.361\times$ compression-only factor in all four models.

Because nvCOMP is CUDA-specific, these Radeon experiments use our native GPU ANS decoder rather than NVIDIA's library. It preserves the same lossless format and pipeline structure, but we do not claim it is tuned to the same degree as nvCOMP. Across sampled real BF16 expert tensors from the four models, the ANS ratio differs from the corresponding synthetic proxy by at most 0.5%. We therefore report the Radeon result as a within-platform portability result, not as an absolute NVIDIA–AMD codec comparison. We make no driver-specific claim for Windows: the

Table 4: Radeon/Windows end-to-end validation under the 8 GB all-miss cap. TBT uses a 256-token context and 100 repetitions; TTFT is paired raw-serial to ANS-full speedup at the 256-token prefill point.

Model	Raw serial TBT (ms)	ANS serial TBT (ms)	ANS full TBT (ms)	TBT gain	TTFT gain
Qwen1.5-MoE-A2.7B	578.2	464.8	341.5	1.69×	1.906×
Gemma4-26B-A4B	853.7	638.8	464.6	1.84×	1.767×
Qwen3.5-35B-A3B	851.0	737.6	436.4	1.95×	2.107×
DeepSeek-V2-Lite	886.5	676.9	491.3	1.80×	1.924×

cross-platform result tests the user-space concurrency and streaming design, with its worker/chunk configuration selected by the same one-time calibration principle.

5.4 Storage Path Utilization

The measurements identify SSD-to-unified-memory delivery as the bottleneck. During an MoE block, unified-memory traffic is only 8.0–9.7% of a 230 GB/s batch-one BF16 weight-streaming reference, although MoE accounts for 91–95% of decoder-layer time. In matched Qwen1.5 runs, ZipMoE realizes 0.46–1.09 GB/s of whole-run effective SSD bandwidth, while our full pipeline realizes 3.84–4.39 GB/s. Whole-run values include non-MoE work and cache hits, which need no SSD read, and therefore differ from the 6.85 GB/s pure-read ceiling. The traffic breakdown and data-path analysis are in Appendix A and Appendix D.

5.5 Interpretation and Scope

The results separate three effects that are often conflated in SSD-backed MoE systems. First, ANS reduces the payload on the slow path only when its GPU decode can be absorbed into the schedule. Second, submitting the routed group together increases the useful request concurrency exposed to the OS/NVMe path; it does not assume that the device grants a linear bandwidth increase for every added worker. Third, hierarchical streaming removes the remaining whole-expert barrier: a ready block can begin Gate/Up computation before all Down weights have arrived. Table 2 isolates the latter two stages under a fixed ANS representation, while the full ladders in Appendix B include the raw serial reference and show how the effects compose.

XSTREAM changes neither routing decisions nor model parameters. The shared expert remains resident, and every routed expert is reconstructed losslessly before its numerical contribution is accumulated. The reported gains are therefore execution improvements, not a trade-off against model accuracy. Their absolute value depends on the model's selected-expert volume, compression ratio, cache regime, storage device, OS activity, and the available CPU/GPU resources. We consequently use one-time, profile-specific calibration and report within-platform comparisons; the agreement across Linux/GB10 and Windows/Radeon supports the design principle. The deployment-specific calibration and run-to-run variability are documented in Appendix A.

6 Conclusion

XSTREAM concurrently submits routed SSD misses, calibrates chunk size and I/O parallelism per deployment profile, and streams losslessly reconstructed blocks through Gate/Up–SwiGLU–Down dependencies. It changes neither routing decisions nor weights.

On GB10, the full pipeline improves throughput by 1.47–$1.53\times$ over the serial no-pipeline schedule and provides 2.07–$4.65\times$ higher matched-memory throughput than ZipMoE. On Radeon/Windows, it reduces end-to-end TBT by 1.69–$1.95\times$ and TTFT by 1.72–$2.13\times$ under an all-miss memory cap. The approach complements caching: hits need no SSD I/O, while misses retain the same concurrent, readiness-driven schedule. The model-, cache-, storage-, OS-, and workload-dependent optimum motivates one-time deployment calibration rather than a universal setting.

AI USE STATEMENT

The research question, core system design, mathematical claims, and proofs were developed by the authors. Generative AI tools assisted in locating and summarizing related work; drafting and editing manuscript text; improving grammar; and copyediting and typesetting the authors' mathematical exposition and proofs. They also assisted with analysis, plotting, and verification scripts. The authors checked the cited sources and wrote the final related-work descriptions, reviewed AI-assisted text and code against the underlying results, and take responsibility for all claims, experiments, and content of this paper.

ETHICS STATEMENT

This work evaluates inference-system performance using pretrained models and existing ShareGPT prompts; it does not recruit participants or collect new user interactions. We report aggregate performance rather than conversation contents. Our runtime preserves model weights and routing, so it does not address biases, unsafe outputs, or privacy risks inherited from the models and prompt data. Faster on-device inference may broaden both beneficial and harmful uses of these models; deployments should retain appropriate model- and application-level safeguards.

REPRODUCIBILITY STATEMENT

Section 4.3 describes the calibration procedure, and Section 5 specifies the evaluation setup. The appendices detail the platform and workload configurations, measurement procedures, calibration sweeps, schedule ablations, and codec comparisons. We will release the code upon acceptance.

REFERENCES

ASUS. ROG Flow Z13 (2025) GZ302: Technical specifications. `https://rog.asus.com/us/laptops/rog-flow/rog-flow-z13-2025/spec/`, 2025. Accessed: 2026-09-22.

Damai Dai, Chengqi Deng, Chenggang Zhao, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 1280–1297, 2024.

Jarek Duda. Asymmetric numeral systems: Entropy coding combining speed of Huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

Zhiyuan Fang, Yuegui Huang, Zicong Hong, Yufeng Lyu, Wuhui Chen, Yue Yu, Fan Yu, and Zibin Zheng. Klotski: Efficient mixture-of-expert inference via expert-aware multi-batch pipeline. *arXiv preprint arXiv:2502.06888*, 2025.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Vincenzo Fornaro and Colibrì contributors. Colibrì: Tiny engine, immense model. `https://github.com/JustVugg/colibri/tree/ce370e87d7b623d7759b52ec2007d75fc5b0e87e`, 2026. Open-source software, commit `ce370e8`, accessed September 2026.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. *arXiv preprint arXiv:2211.15841*, 2023.

Xin He, Shunkang Zhang, Kaijie Tang, Shaohuai Shi, Yuxin Wang, Zihao Zeng, Zhenheng Tang, Xiaowen Chu, Haiyan Yin, Ivor W. Tsang, and Yew Soon Ong. Expertflow: Efficient mixture-of-experts inference via predictive expert caching and token scheduling. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference (DAC)*, 2026. doi: 10.1145/3770743.3804292.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, Joe Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. *arXiv preprint arXiv:2206.03382*, 2022.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

Kanchan Joshi, Anuj Gupta, Javier Gonzalez, Ankit Kumar, Krishna Kanth Reddy, Arun George, Simon Lund, and Jens Axboe. I/o passthru: Upstreaming a flexible and efficient i/o path in linux. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*, pp. 107–121, 2024.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations (ICLR)*, 2025.

Michael Kerrisk. open(2): Open and possibly create a file. Linux man-pages, `https://man7.org/linux/man-pages/man2/open.2.html`, 2026a. Accessed: 2026-09-25.

Michael Kerrisk. pread(2): Read from a file descriptor at a given offset. Linux man-pages, `https://man7.org/linux/man-pages/man2/pread.2.html`, 2026b. Accessed: 2026-09-25.

Gyusun Lee, Seokha Shin, Wonsuk Song, Tae Jun Ham, Jae W. Lee, and Jinkyu Jeong. Asynchronous i/o stack: A low-latency kernel i/o stack for ultra-low latency ssds. In *2019 USENIX Annual Technical Conference*, 2019.

Linux Kernel Documentation. Multi-queue block i/o queueing mechanism (blk-mq). `https://docs.kernel.org/block/blk-mq.html`, 2026. Accessed: 2026-09-21.

Microsoft. Overlapped operations. Microsoft Learn, `https://learn.microsoft.com/en-us/windows/win32/devio/overlapped-operations`, 2021. Accessed: 2026-09-25.

Microsoft. File buffering. Microsoft Learn, `https://learn.microsoft.com/en-us/windows/win32/fileio/file-buffering`, 2025a. Accessed: 2026-09-25.

Microsoft. Readfile function. Microsoft Learn, `https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-readfile`, 2025b. Accessed: 2026-09-25.

MoE-Infinity Contributors. Moe-infinity: Source repository and documentation. `https://github.com/EfficientMoE/MoE-Infinity`, 2026. Accessed: 2026-09-25.

NVIDIA Corporation. NVIDIA nvCOMP documentation. `https://docs.nvidia.com/cuda/nvcomp/`, 2025. Accessed September 2026.

NVM Express, Inc. Nvm express base specification, revision 2.0d. `https://nvmexpress.org/wp-content/uploads/NVM-Express-Base-Specification-2.0d-2024.01.11-Ratified.pdf`, 2024.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

Ying Sheng, Lianmin Zheng, Binhang Yuan, et al. Flexgen: High-throughput generative inference of large language models with a single gpu. *arXiv preprint arXiv:2303.06865*, 2023.

Andrii Skliar, Ties van Rozendaal, Romain Lepert, Todor Boinovski, Mart van Baalen, Markus Nagel, Paul N. Whatmough, and Babak Ehteshami Bejnordi. Mixture of cache-conditional experts for efficient mobile device inference. *Transactions on Machine Learning Research*, 2025.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. Moe-infinity: Offloading-efficient moe model serving. *arXiv preprint arXiv:2401.14361*, 2024.

Yuchen Yang, Yaru Zhao, Pu Yang, Shaowei Wang, and Zhi-Hua Zhou. Zipmoe: Efficient on-device moe serving via lossless compression and cache-affinity scheduling. In *Proceedings of the 43rd International Conference on Machine Learning (ICML)*, 2026.

Shuzhang Zhong, Ling Liang, Yuan Wang, Runsheng Wang, Ru Huang, and Meng Li. Adapmoe: Adaptive sensitivity-based expert gating and management for efficient moe inference. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024. doi: 10.1145/3676536.3676741.

A Extended Evaluation Details

Reproducibility and deployment. The calibration is performed once for each *model–platform–memory- budget* profile before serving: it selects the chunk size and I/O-worker operating point using measured loading diagnostics and complete-model validation, then reuses that entry for subsequent requests. This cost is not charged to per-token latency. The choice depends on selected-expert volume, ANS ratio, GPU decode batch shape, the SSD path, and the OS storage stack; it should therefore not be treated as a universal constant. The selected operating point is validated with complete model execution, not a read-only microbenchmark.

Exact wall-clock values on a storage path can vary with OS wakeups, background activity, device firmware state, and cache pressure. We address this by reporting repeated runs, paired/interleaved comparisons where both systems are available, and tail statistics for the Radeon all-miss TBT experiment. The main tables report end-to-end TTFT and TBT; detailed repetitions, confidence intervals, per-layer calibration sweeps, and the full schedule ladders are provided in the appendix.

Numerical validation. We tested two distinct properties. First, ANS reconstruction was bitwise identical to the original BF16 weights for all 736 tested experts. Second, within the same patched Qwen1.5 implementation and prompt, we changed only the execution mode among serial, whole-expert barrier, and full pipeline. Across 16 generated tokens, each pair of modes had bitwise-identical logits in all 1,519,360 compared elements (maximum absolute difference zero), and generated the same tokens. The implementation sorts completed expert results by expert ID before its `index_add_` scatter, so expert arrival order does not determine that reduction order. This experiment verifies schedule equivalence for the tested run; it does not claim bitwise equivalence to a different implementation or establish equality for every input and model.

Cross-platform SSD read path. An I/O worker is a host thread that submits reads for expert chunks; it is distinct from a CPU decompression worker. On GB10/Linux, each worker issues blocking positional `pread(2)` reads on an `O_DIRECT` file descriptor. Positional means that the request specifies its file offset; `O_DIRECT` minimizes page-cache effects (Kerrisk, 2026b;a). A worker has at most one application read in flight. On Radeon/Windows, a worker calls `ReadFile` with an `OVERLAPPED` request on a handle opened with `FILE_FLAG_NO_BUFFERING` and `FILE_FLAG_OVERLAPPED`, then waits for that request's completion before its next read (Microsoft, 2025b; 2021; 2025a). Thus requests from different workers can be outstanding on either platform; the Windows implementation does not use an I/O completion port.

NVMe is the SSD command interface, and the operating system and storage controller queue submitted requests before they complete. Linux uses the multi-queue block layer (`blk-mq`) on this path; Windows uses its own storage stack. One 2 MB application read may be split into multiple block requests and NVMe commands, so the number of workers is neither the measured device queue depth nor the number of NVMe commands (Linux Kernel Documentation, 2026; NVM Express, Inc., 2024). The GB10 `aqu-sz` reported in the main text is a measured average number of outstanding block requests, not a count of application threads. Request granularity and offered concurrency jointly affect the realized bandwidth (Lee et al., 2019; Joshi et al., 2024).

This appendix records the evaluation configuration and the complete cross-platform measurements that are summarized in Sec. 5. The GB10 experiments use real model checkpoints and matched 8–16 GB runtime budgets. The Radeon/Windows experiments use the same model geometries under an 8 GB all-miss cap: after an expert is consumed, it is replaced by an expert needed by a later layer, so a cache entry cannot serve a subsequent request.

Models and platforms. Table 5 lists the sparse FFN geometries that determine the routed payload. Table 6 distinguishes the two OS/device stacks. A reported I/O-worker count means a host submission worker, not a GPU kernel or a CPU codec worker. Both systems use a PCIe 4.0 $\times 4$ NVMe

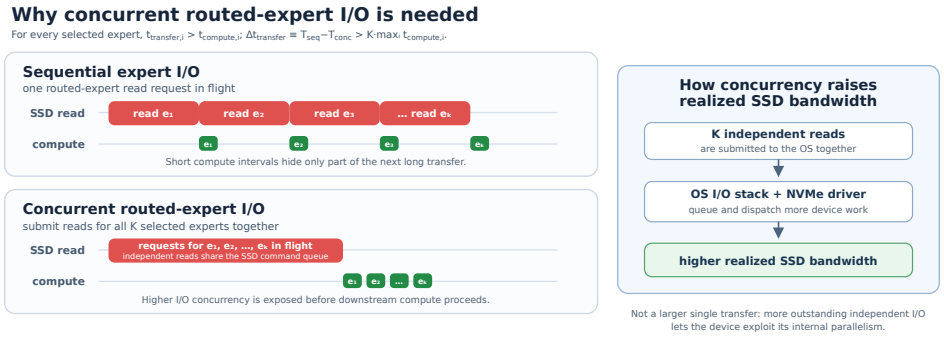

Figure 4: Sequential versus concurrent routed-expert I/O. Concurrent submission creates multiple independent outstanding requests; it does not partition a single transfer into artificial per-expert slices.

Table 6: Evaluation platforms and runtime caps.

System / OS	CPU; GPU	Unified memory	SSD path; runtime cap
GB10 / Ubuntu 24.04.3	20-core Arm; Blackwell	119 GiB LPDDR5X	Gen4 $\times 4$ NVMe; 8/12/16 GB caps
Radeon 8060S / Win 11 Pro 25H2	Ryzen AI Max+ 395; Radeon 8060S	64 GiB LPDDR5X-8000	Gen4 $\times 4$ NVMe; 8 GB all miss

path; the runtime budget is deliberately smaller than physical unified memory so that the evaluation measures expert misses rather than a fully resident model.

End-to-end protocol. For Radeon TBT, each cell uses a 256-token context and 100 all-miss repetitions. Table 7 reports the mean and tail latency of the full ANS operator-aware schedule against raw serial. TTFT is measured over three prompt lengths; Table 10 reports paired speedup and bootstrap confidence intervals, with $n = 100$ at 256 tokens and $n = 20$ at the bracketing prompt lengths. Table 8 gives the corresponding absolute times. Table 9 retains the ANS-serial intermediate, so that all reported Radeon schedule arms are visible. These are end-to-end model measurements, not per-layer load times.

GB10 traffic profile. Table 11(a) shows that MoE execution dominates the per-layer time while unified-memory traffic remains far below the 230 GB/s reference, identifying storage delivery rather than unified-memory bandwidth as the constrained path. Panel (b) fixes ZipMoE `prefetcher_topk=4`; the matched-memory figure in the main paper chooses its better documented setting at each budget and therefore answers a different question. Figure 4 illustrates the separate scheduling distinction: submitting K independent routed-expert reads exposes useful request concurrency to the OS/NVMe path, whereas a one-expert-at-a-time schedule repeatedly drains that opportunity at each expert boundary.

Table 5: Evaluated MoE models.

Model	Total / active	Top-K / experts	Shared	Hidden / intermediate
Qwen1.5	14.3B / 2.7B	4 / 60	Yes	2048 / 1408
DeepSeek-V2-Lite	15.7B / 2.4B	6 / 64	Yes	2048 / 1408
Gemma4-26B-A4B	26B / 4B	8 / 128	Yes	2816 / 704
Qwen3.5-35B-A3B	35B / 3B	8 / 256	Yes	2048 / 512

On GB10, Ubuntu runs kernel 6.14.0-1015-nvidia on aarch64; the 20 Arm cores combine Cortex-X925 and Cortex-A725 cores in one NUMA node. The integrated GB10 GPU has compute capability 12.1, NVIDIA driver 580.126.09, and CUDA toolkit 13.1. Its 4 TB NVMe drive is model `ESL04TBTLCZ-27J4-TYN` (firmware `ERFM12.0`). The approximately 128.5 GB reported by `cudaMemGetInfo` and the OS-reported 119 GiB describe the same CPU/GPU shared physical pool, not additive capacities.

Table 7: Radeon/Windows end-to-end TBT (ms), 256-token context, all miss ($n = 100$).

Model	Raw serial mean	ANS full mean	ANS full p95	ANS full p99	Gain
Qwen1.5-MoE-A2.7B	578.2	341.5	342.9	351.3	1.69×
Gemma4-26B-A4B	853.7	464.6	470.6	573.9	1.84×
Qwen3.5-35B-A3B	851.0	436.4	439.5	444.4	1.95×
DeepSeek-V2-Lite	886.5	491.3	493.7	506.0	1.80×

Table 8: Radeon/Windows absolute TTFT (s): raw serial and ANS full.

Model	Arm	16 tokens ($n = 20$)	256 tokens ($n = 100$)	1,024 tokens ($n = 20$)
Qwen1.5	Raw serial	5.340 [5.331, 5.349]	8.045 [8.037, 8.053]	8.441 [8.430, 8.452]
	ANS full	2.809 [2.805, 2.812]	4.222 [4.219, 4.225]	4.564 [4.560, 4.569]
Gemma4	Raw serial	5.629 [5.618, 5.640]	6.676 [6.654, 6.699]	7.023 [7.010, 7.037]
	ANS full	3.271 [3.268, 3.273]	3.778 [3.773, 3.784]	4.059 [4.054, 4.064]
Qwen3.5	Raw serial	5.343 [5.333, 5.354]	6.154 [6.150, 6.159]	6.459 [6.446, 6.472]
	ANS full	2.514 [2.511, 2.518]	2.922 [2.919, 2.924]	3.337 [3.331, 3.344]
DeepSeek	Raw serial	7.630 [7.623, 7.639]	9.358 [9.354, 9.362]	9.675 [9.668, 9.683]
	ANS full	3.968 [3.965, 3.971]	4.864 [4.862, 4.865]	5.132 [5.129, 5.135]

The model table makes the workload distinction explicit: the routed set ranges from top-4 of 60 experts to top-8 of 256 experts, while the selected per-layer payload also varies substantially. The same scheduling mechanism is therefore evaluated across different group sizes and I/O volumes rather than only one favorable MoE shape. The platform table records the deliberately capacity-limited regime: physical unified memory is not used as an implicit fully-resident cache.

Radeon TBT interpretation. Table 7 shows that the full lossless schedule reduces all-miss end-to-end TBT on every model, not merely the per-layer transfer. The $1.69\times$–$1.95\times$ gains include the full decoding path. The p95 and p99 columns keep this claim tied to tail behavior rather than mean latency alone; the detailed distribution is intentionally retained here because the main paper reports only the compact cross-platform summary.

TTFT interpretation. The absolute values in Table 8 show that the improvement persists from short prompts through the 256-token saturated point and the 1,024-token bracket. The performance benefit is therefore not an artifact of reporting a single prompt length. The paired ratios and their bootstrap intervals are given next so that the comparison is not inferred from rounded absolute means.

Separating compression from scheduling. Table 9 supplies the intermediate ANS-serial arm. ANS lowers the transfer payload, but the full schedule then removes the remaining expert-level waiting and produces the reported final TBT. Thus the raw-to-full gains in Table 7 are composite system gains, whereas the serial-to-full difference isolates the scheduling contribution at a fixed lossless representation.

Table 10: Radeon/Windows paired TTFT speedup, raw serial to ANS full operator-aware.

Model	16 tokens ($n = 20$)	256 tokens ($n = 100$)	1,024 tokens ($n = 20$)
Qwen1.5-MoE-A2.7B	1.901× [1.898, 1.904]	1.906× [1.903, 1.908]	1.849× [1.846, 1.853]
Gemma4-26B-A4B	1.721× [1.718, 1.725]	1.767× [1.762, 1.771]	1.730× [1.726, 1.734]
Qwen3.5-35B-A3B	2.125× [2.120, 2.130]	2.107× [2.104, 2.109]	1.935× [1.931, 1.941]
DeepSeek-V2-Lite	1.923× [1.921, 1.925]	1.924× [1.923, 1.925]	1.885× [1.883, 1.887]

The intervals in Table 10 are narrow under the paired/interleaved protocol. They confirm that the 256-token TTFT reductions reported in the main text, as well as the bracketing prompt-length results, are stable relative to run-to-run storage-path variation.

Table 9: Radeon/Windows schedule decomposition: end-to-end TBT (ms), 256-token context ($n = 100$).

Model	Raw serial	ANS serial	ANS full operator-aware
Qwen1.5-MoE-A2.7B	578.2	464.8	341.5
Gemma4-26B-A4B	853.7	638.8	464.6
Qwen3.5-35B-A3B	851.0	737.6	436.4
DeepSeek-V2-Lite	886.5	676.9	491.3

Table 11: GB10 unified-memory traffic and controlled SSD profile.

(a) Unified-memory traffic during the MoE block

Model	MoE/layer	UM traffic of 230 (GB/s)	GB/s
Qwen1.5-MoE	93.3%	18.4	8.0%
DeepSeek-V2-Lite	94.7%	22.4	9.7%
Gemma4-26B	91.2%	20.8	9.0%
Qwen3.5-35B	91.3%	19.7	8.6%

(b) Qwen1.5 matched-memory SSD profile (64 output tokens; $n = 3$)

Engine	Budget	Tok/s	Wall (s)	Read (GB)	GB/s	Peak
ZipMoE	8 GB	1.87 ± 0.17	34.44	37.53	1.09	14%
ZipMoE	12 GB	3.15 ± 0.32	20.48	9.34	0.46	6%
Ours (full)	8 GB	5.50 ± 0.02	11.64	51.15	4.39	56%
Ours (full)	12 GB	7.18 ± 0.02	8.91	34.20	3.84	49%
Pure-read ceiling	–	–	–	–	6.85	87%

Reading the storage profile. The left panel of Table 11 rules out unified-memory bandwidth as the dominant MoE bottleneck in these runs: its traffic uses only 8.0–9.7% of the 230 GB/s reference while MoE occupies 91–95% of layer time. On the right, whole-run SSD throughput is lower than the pure-read ceiling because non-MoE work and cache hits do not issue SSD reads. Under the same memory caps, our pipeline keeps the drive busy for a much larger fraction of execution than ZipMoE; the latter's lower byte volume at 12 GB does not by itself translate into a lower wall time.

GPU compute-time proxy in the serial storage-bound path. In a separate no-cache Qwen1.5-MoE-A2.7B serial profile, a token takes 282.2 ms, of which 267.0 ms (95%) is spent in MoE blocks. The MoE phase reads at 3.90 GB/s from SSD while moving 20.9 GB/s through unified memory, only 9% of the 230 GB/s reference; non-MoE execution issues no SSD reads and reaches 224.4 GB/s of memory traffic. The fully resident MoE compute reference is 28.2 ms, giving $28.2/267.0 = 10.6\%$ of the SSD-backed MoE interval and leaving 89.4% outside resident-equivalent expert computation. With an 8 GB cache, MoE time is 184.7 ms, so the corresponding non-compute fraction is $1 - 28.2/184.7 = 84.7\%$. These fractions quantify the exposed data-path and scheduling time in this serial profile; they are not GPU hardware-idle counters, and the residual need not consist entirely of SSD waits.

B STEPWISE GB10 SCHEDULE ABLATIONS

These ladders are fixed-$N = 20$ implementation diagnostics, not replications of the calibrated $N = 8$ end-to-end comparison in Table 2. They hold $C = 1$ MB, a 12 GB budget, ShareGPT-256 prompts, 64 output tokens, and the same LRU policy. Rung 1 is raw serial with one worker; rung 2 adds ANS but retains that one-worker, no-pipeline structure; rung 3 submits the selected group through one I/O worker; rung 4 increases the pool to 20 workers; rungs 5–7 then add ANS, chunk-level pipelining, and the operator-aware schedule. Thus, the ladders separate the implementation steps without claiming that their values should numerically reproduce Table 2.

"Multi I/O" denotes the 20-worker pool. A one-worker rung uses one I/O submission thread total. ANS is lossless (Duda, 2013); its decode uses the GPU-batched path described in Sec. 4.2. The two model ladders establish that the ordering of benefits is not specific to Qwen1.5, while the following I/O ablation separates inter-expert concurrency from threading chunks within one expert.

Reading the ladders. The "Step" column is the throughput ratio to the immediately preceding rung; the line below each table reports the cumulative raw-serial-to-full change. Rung 3 is a group-submission control at one worker, while rung 4 isolates a larger offered I/O pool. With the representation then fixed, rung 5 measures ANS without intra-expert streaming, rung 6 enables block readiness, and rung 7 changes only the operator schedule. These diagnostic rows expose which implementation dependency each step removes; they are not a second end-to-end benchmark.

For Qwen1.5, the raw one-worker ANS row is slower than raw because its decode cost is exposed without enough I/O overlap. Once the routed group and the worker pool are exposed (rungs 4–5), the lower payload becomes useful; chunk readiness (rung 6) and the operator-aware Down schedule (rung 7) then reduce both TTFT and TBT further. This sequence is why the paper does not present compression as a standalone storage optimization.

DeepSeek follows the same ordering despite its different top-K and selected payload. In particular, the group-level I/O step and the ANS pipeline provide the larger transitions, while the final operator-aware stage is a smaller but consistent refinement after the data path has already been made concurrent. The agreement between the two ladders supports the staged explanation used in the main paper rather than a Qwen-specific artifact.

Repeated schedule check. Table 14(a) compares all four three-run point estimates with separate 20-run 95% speedup intervals; every point estimate lies within its corresponding interval. Panel (b) gives mean $\pm$ standard deviation for each rerun arm available for Qwen1.5 and DeepSeek. The intervals belong to the reruns, not the three-run cells; Gemma4 and Qwen3.5 have no supplied per-arm variance. The rerun gain need not equal the ratio of rounded arm means.

Inter-expert concurrency beyond intra-expert threading. Table 15 holds Qwen1.5's selected-expert payload fixed while varying the I/O thread pool. The single-expert column parallelizes chunks only within one expert, so experts still transfer in serial order. The four-expert column submits the routed group together. Group-level payload rate is higher at every measured thread count: at ten threads, 5.60 versus 4.64 GB/s. The grouped transfer completes in 12.36 ms rather than the 14.92 ms obtained by serially repeating the 3.73 ms single-expert transfer.

The comparison shows why intra-expert threading is not a substitute for inter-expert concurrency. Both columns increase the offered work as the thread count rises, but only the four-expert group maintains multiple independent expert requests in flight. The improvement is therefore attributable to the request structure exposed to the OS/NVMe path, not to assigning a fixed quarter of a transfer to each expert.

Batch scaling. Table 16 varies the decode batch while holding the Qwen1.5 runtime configuration fixed. All rows use lossless ANS, $N = 8$ I/O workers, $C = 1$ MB, a 12 GB (22-slot) cap, ShareGPT-256 prompts, 64 generated tokens, and the same LRU policy ($n = 3$). "Serial" processes one routed expert at a time; "concurrent" submits the routed group together. The final column compares the full design directly with the serial no-pipeline arm.

The full-over-serial gain rises from $1.517\times$ at batch one to $2.401\times$ at batch eight, then changes with cache pressure and the routed union at batch 16. We consequently treat the table as evidence that

Table 12: Qwen1.5-MoE-A2.7B full ablation ladder ($n = 3$).

#	Rung	Multi I/O	ANS	ANS pipeline	Op-aware	Tok/s	Step	TTFT (s)	TBT (s)	Decode tok/s
1	Raw serial, 1 worker	–	–	–	–	2.666	–	6.287	0.2815	3.555
2	+ ANS, no pipeline	–	Y	–	–	2.563	0.961×	6.414	0.2948	3.396
3	Raw concurrent ($K = 4$), 1 I/O worker	–	–	–	–	2.673	1.003×	6.295	0.2804	3.569
4	+ 20 I/O workers	Y	–	–	–	3.953	1.479×	3.989	0.1937	5.164
5	+ ANS, no pipeline	Y	Y	–	–	4.851	1.227×	2.890	0.1636	6.112
6	+ ANS pipeline	Y	Y	Y	–	5.476	1.129×	2.523	0.1455	6.873
7	+ operator-aware (full)	Y	Y	Y	Y	5.607	1.024×	2.417	0.1428	7.002

Cumulative 1→7: throughput 2.10×, TTFT 2.60×, TBT 1.97×, and decode throughput 1.97×.

Table 13: DeepSeek-V2-Lite full ablation ladder ($n = 3$).

#	Rung	Multi I/O	ANS	ANS pipeline	Op-aware	Tok/s	Step	TTFT (s)	TBT (s)	Decode tok/s
1	Raw serial, 1 worker	–	–	–	–	2.503	–	6.841	0.2979	3.369
2	+ ANS, no pipeline	–	Y	–	–	2.215	0.885×	7.688	0.3371	2.968
3	Raw concurrent ($K = 6$), 1 I/O worker	–	–	–	–	2.592	1.036×	6.424	0.2906	3.452
4	+ 20 I/O workers	Y	–	–	–	3.548	1.369×	4.159	0.2206	4.537
5	+ ANS, no pipeline	Y	Y	–	–	4.117	1.160×	3.647	0.1889	5.296
6	+ ANS pipeline	Y	Y	Y	–	4.617	1.121×	3.014	0.1722	5.808
7	+ operator-aware (full)	Y	Y	Y	Y	4.746	1.028×	2.899	0.1680	5.951

Cumulative 1→7: throughput 1.90×, TTFT 2.36×, TBT 1.77×, and decode throughput 1.77×.

batching creates more opportunity for inter-expert concurrency, not as a claim of one universal best batch size. The full schedule is higher than the serial no-pipeline reference at every measured batch.

Table 14: GB10 schedule uncertainty: main estimates and separate reruns.

(a) Full / serial no-pipeline speedup

Model	Main gain ($n = 3$)	Rerun 95% CI ($n = 20$)
Qwen1.5	1.473×	[1.4465, 1.5167]
DeepSeek	1.526×	[1.5071, 1.5584]
Gemma4	1.486×	[1.4262, 1.4929]
Qwen3.5	1.528×	[1.5011, 1.5341]

(b) Rerun throughput (tokens/s; $n = 20$, mean $\pm$ sd)

Model	Serial no pipeline	ANS serial	Concurrent no pipeline	Full operator-aware	Rerun gain
Qwen1.5	4.098 ± 0.478	4.801 ± 0.444	4.925 ± 0.480	6.046 ± 0.493	1.482×
DeepSeek	3.237 ± 0.309	3.764 ± 0.315	3.946 ± 0.322	4.951 ± 0.370	1.533×

Table 15: GB10 Qwen1.5: threaded single expert versus four-expert group.

I/O threads	Single ms	Single GB/s	4 experts ms	4 experts GB/s
4	4.47	3.87	14.68	4.71
5	3.91	4.42	13.40	5.16
6	3.86	4.49	13.31	5.20
7	3.92	4.41	13.07	5.30
8	3.84	4.51	12.58	5.50
9	3.87	4.47	12.43	5.57
10	**3.73**	**4.64**	**12.36**	**5.60**
11	3.75	4.62	12.70	5.45
12	3.83	4.52	12.73	5.44

Table 16: GB10 Qwen1.5 batch ablation (tokens/s).

Batch	Serial no pipeline	Serial pipeline	Concurrent no pipeline	Concurrent pipeline	Full	Full / serial no pipeline
1	3.900	4.706	4.771	5.684	5.917	1.517×
2	4.656	5.673	5.249	7.427	7.824	1.680×
4	6.178	8.052	8.437	10.980	11.644	1.900×
8	4.102	5.199	7.265	9.238	9.849	**2.401**×
16	5.310	6.544	7.503	9.329	9.824	1.850×

C WHY CHUNK SIZE AND WORKER COUNT REQUIRE JOINT CALIBRATION

This appendix formalizes why the chunk size C and the number of I/O submission workers N in Sec. 4.3 are tuned jointly. The result is intentionally a statement about the absence of a *general guarantee* for independent one-dimensional sweeps: it does not claim that every non-separable system must exhibit a large measurable shift in its optimum at every operating point.

C.1 WHEN INDEPENDENT ONE-DIMENSIONAL SWEEPS ARE EXACT

Let $F(x, y)$ be a two-parameter objective. Suppose that a practitioner fixes reference values y_0 and x_0, independently computes

$$\hat{x} \in \arg\min_x F(x, y_0), \qquad \hat{y} \in \arg\min_y F(x_0, y), \tag{14}$$

and reports $(\hat{x}, \hat{y})$. This procedure is guaranteed to work for arbitrary reference values only when the best response of each coordinate is invariant to the other coordinate. A convenient sufficient condition is additive separability,

$$F(x, y) = G(x) + H(y), \tag{15}$$

because then $\arg\min_x F(x, y) = \arg\min_x G(x)$ for every y, and likewise for y. Additive separability is sufficient, not necessary: non-separable objectives can happen to share the same optimum. Conversely, non-separability removes the guarantee that the two independent searches will return a joint optimum.

For example, let $F(x, y) = (x - y)^2 + \frac{1}{2}x$ on $[0, 2]^2$. Its unique joint minimum is $(0, 0)$ with value zero. If the x-sweep is instead run at $y_0 = 2$, it returns $\hat{x} = 7/4$; if the y-sweep is run at $x_0 = 0$, it returns $\hat{y} = 0$. The independently assembled pair $(7/4, 0)$ has value $63/16$, and is therefore not jointly optimal. The cross term $-2xy$ in the expanded objective makes each coordinate's best response depend on the other.

C.2 INTERACTION IN THE SSD-BACKED MOE PATH

Why worker count affects realized bandwidth. A worker does not receive additional bandwidth merely by existing. Instead, each worker submits independent chunk reads; the OS I/O stack and NVMe driver then have more outstanding operations that can be placed in software and controller command queues. At low concurrency, this additional queue occupancy can expose more of the SSD's internal parallelism. The response is inherently non-linear: it saturates once the software queues, controller queues, flash channels, or host-memory path are sufficiently occupied, and it can even regress when CPU decompression and memory traffic contend with I/O. Consequently, N is a control on the amount of offered I/O concurrency, not a claim that the OS grants bandwidth linearly in N.

For one routed-expert payload of fixed uncompressed size R, consider the following local model of the critical path:

$$\widetilde{T}(C, N) = \frac{R}{\rho(C)B(N)} + D(C). \tag{16}$$

Here $\rho(C)$ is the lossless compression ratio at chunk size C, and $B(N)$ is the realized storage-to-unified-memory bandwidth induced by the OS/NVMe path at worker count N. The term $D(C)$ collects chunk-dependent non-I/O costs, including decompression, reconstruction, and the loss of streamability at overly large chunks. The full runtime is more coupled than this reduced model: request granularity affects queueing and driver behavior, so the effective bandwidth is generally $B(C, N)$; decompression and shared-memory contention can likewise contribute a term $A(C, N)$. Equation 16 deliberately omits these additional interactions. It is therefore sufficient for showing why a one-dimensional guarantee is unavailable, while the real implementation is selected from measured end-to-end latency.

For twice-differentiable continuous relaxations of C and N, additive separability requires a zero mixed partial derivative. In contrast, Eq. 16 gives

$$\frac{\partial^2 \widetilde{T}}{\partial C\, \partial N} = \frac{R\, \rho'(C) B'(N)}{\rho(C)^2 B(N)^2}. \tag{17}$$

Whenever compression ratio and achieved bandwidth both change locally with their respective controls, this quantity is nonzero. Thus, in the unsaturated OS/NVMe regime where additional outstanding reads still increase realized bandwidth, the storage term cannot be written as a sum of a chunk-only cost and a worker-only cost. The omitted $B(C, N)$ and $A(C, N)$ terms can preserve or add further coupling; they are not needed for the argument.

Equation 17 establishes coupling only while $B'(N)$ is nonzero; it does not provide a universal directional prediction for how an optimal chunk size moves with N. In particular, the full runtime includes the omitted $B(C, N)$ and $A(C, N)$ terms, and the finite candidate set need not have an interior differentiable minimizer. We therefore select (C, N) from measured end-to-end latencies rather than infer a direction for $C^\star(N)$ from the reduced model.

C.3 Connection to the measured search

Figure 2 provides the empirical counterpart to this argument for selected-expert loading, excluding expert computation. The independently assembled chunk-only and worker-only selections reach 8.635 ms, while the best joint point, $C = 1$ MB and $N = 20$, reaches 8.274 ms. This 0.361 ms (4.18%) difference uses absolute latency; it is not obtained by subtracting heatmap ratios with different raw denominators. A decode-inclusive sweep separately measures the two stages over the same grid. Its reported bandwidth is the payload-stage read bandwidth, excluding end-to-end GPU decode and computation. Realized SSD bandwidth is nearly flat in C and rises from 5.2–5.9 GB/s at $N = 1$ to about 6.5 GB/s at $N = 2$, after which it remains saturated through $N = 20$. In contrast, GPU decompression is strongly chunk-dependent: small chunks are launch-overhead dominated, while overly large chunks expose too little batch parallelism. Its variation with N is weak and non-monotonic.

This measured saturation explains the broad high-performance ridge. Beyond the low-concurrency transition, $B'(N) \approx 0$, so the reduced-model mixed interaction in Eq. 17 becomes negligible and no monotonic $C^\star(N)$ trend is expected. Indeed, the selected chunk sizes vary non-monotonically within the near-optimal region. A flat ridge therefore does not restore a one-dimensional optimality guarantee; it means that several nearby joint configurations have similar measured latency.

C.4 Eight-expert sensitivity and concurrent-stage timing

In a separate Qwen1.5-MoE-A2.7B batch-two sensitivity test, we used the chunk-size/worker setting optimized to fetch four experts to fetch eight experts, then compared it against a setting optimized to fetch eight experts. Both settings thus read the same eight experts. Their loading times differ by only 0.15% in this test. For batch-one all-miss loading, prompt length does not change Qwen1.5's top-K count or the encoded size of each routed expert. A change in the number of distinct experts fetched under batching also has a negligible measured penalty within the four-to-eight-expert range tested here. This is an offline sensitivity check, not an inference-time change of C; it uses a separate fetch experiment from the all-miss diagnostic in Fig. 2. Cached and full-model workloads may have different operating points.

To check whether simultaneous reads materially delay GPU decode on GB10, we also randomized measurement order at $C = 1$ MB and $N = 8$. Relative to the isolated stages, the concurrent-read timing ratio was 1.0036 (about 0.4%) and the decode timing ratio was 1.0343, with a paired BCa 95% confidence interval of [0.9983, 1.0802] and Wilcoxon $p = 0.158$ for decode. We detected no statistically significant decode slowdown in this configuration. These are timing measurements: GB10 provides no LPDDR bandwidth counter in this experiment, so they neither directly measure memory-bus contention nor rule out interference at other operating points.

The implication is practical rather than universal: a one-dimensional sweep at a fixed default setting may find a good configuration by chance, but it cannot certify optimality when compression granularity and storage concurrency interact. We consequently use the finite joint calibration of Sec. 4.3 as a one-time selection procedure for each target platform and workload.

Table 17: Matched Qwen1.5 raw and ANS expert-load check (ms).

I/O workers	Raw	ANS SSD	ANS decode	ANS total
1	12.665	8.470	1.833	10.303
20	10.258	7.132	1.157	8.289

D ZipMoE-Style Recovery Path versus GPU-Batched ANS

This appendix isolates the compression backend using the Qwen1.5-MoE-A2.7B selected-expert payload (69.21 MB uncompressed), read by the conventional serial one-expert-ahead schedule on NVIDIA GB10. It makes two distinctions that are easy to conflate. Whenever we report a raw "thread" count, it means an *I/O submission worker*: a worker that issues SSD reads and increases offered I/O concurrency. In contrast, a ZipMoE-style "thread" is a *CPU OS worker* used for entropy decoding and the associated reconstruction workflow; its count is *not* an NVMe queue depth. Our ANS configuration uses I/O workers for reads and performs decode on the GPU, so its decoding work does not contend for the same CPU worker pool.

This appendix is *not* the model-specific joint 2-D calibration of Sec. 4.3. It does not sweep or select an ANS chunk size–I/O-worker pair. Instead, it is a backend-isolation study: the raw one- versus eight-worker rows isolate the effect of parallel SSD-read submission, while the compression rows isolate the cost and benefit of the respective compression/decompression backend at the reported configuration. The jointly calibrated (C, N) operating point remains a separate, model- and platform-dependent deployment optimization. The separate matched-memory cache-budget comparison is reported in Appendix E.

What UMA does and does not remove. ZipMoE's published zero-copy design should not be interpreted as a physical CPU-to-GPU-memory copy: both processors address the same coherent LPDDR pool. For a cache-miss tensor, however, its CPU workers first decode the compressed exponent stream into an intermediate buffer. The subsequent GPU recovery kernel reads that decoded buffer and the uncompressed sign–mantissa stream, then writes a reconstructed BF16 tensor that the expert GEMM consumes. Thus, the relevant cost is not a RAM-to-RAM transfer; it is CPU entropy-decoding work, intermediate-buffer traffic on the shared memory system, the GPU recovery read/write traffic, and the dependency between those stages. Our GPU-batched ANS path still writes its decoded destination tensor, but it removes the CPU-produced exponent intermediate and the separate GPU bit-field-reassembly step.

This comparison does not claim that CPU decompression is intrinsically unsuitable for UMA. A CPU-centric codec can suit a device with underutilized CPU cores, limited GPU decode capability, or no GPU-compatible entropy decoder. On the present shared-memory systems, however, a cache-miss path benefits from avoiding host-side decode and consuming SSD-resident chunks with a GPU-parallel decoder.

D.1 Matched ANS loading check

To verify the lossless-compression contribution independently of the end-to-end scheduler, we rerun raw BF16 and ANS loading in one harness with the same four Qwen1.5 selected experts, file-preparation path, and I/O-worker count. The decoded tensors are byte-exact in every configuration. ANS reduces the selected-expert loading time at both shallow and deeper offered I/O concurrency, while its GPU decode remains a small part of the measured path. This is a codec check, not an implementation-faithful CPU-codec comparison of ZipMoE.

At one I/O worker, ANS reduces the Qwen1.5 selected-expert load from 12.665 to 10.303 ms; at the deeper offered-I/O setting it reduces the load from 10.258 to 8.289 ms. The SSD-transfer reduction is larger than the GPU-decode addition in both rows. This confirms the basic codec trade-off used by the pipeline; the schedule and cache experiments elsewhere in the paper determine whether that local reduction becomes an end-to-end gain.

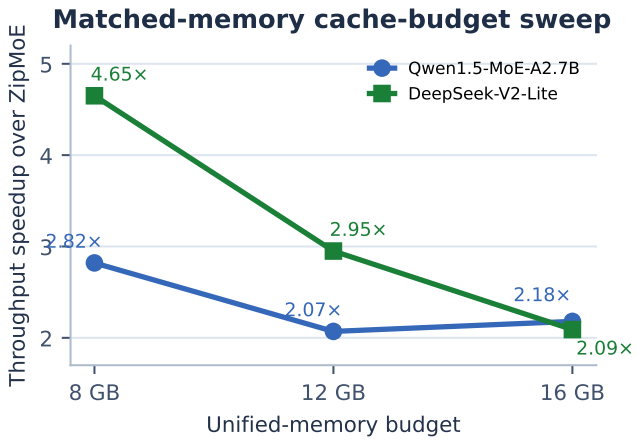

Figure 5: Matched-memory throughput speedup over ZipMoE.

E CACHE-BUDGET SENSITIVITY

ZipMoE is principally designed to reduce fetches through hierarchical caching and cache-affinity scheduling (Yang et al., 2026). Those mechanisms are valuable when an expert remains resident, but a bounded unified-memory budget cannot retain every expert for all requests. This appendix therefore studies the complementary cache-miss regime, in which selected experts must be read from SSD and the transfer path is unavoidable.

The two cache designs should not be equated by a single hit-rate number. Our fixed-slot cache stores a fully decoded expert, so a hit is immediately executable. ZipMoE's hierarchical bit-field cache can retain part of an expert representation; partial residency may reduce fetched bytes, but the complementary data and recovery are still required before the BF16 tensor is ready for the GPU (Yang et al., 2026). We therefore compare end-to-end time, SSD bytes, and realized whole-run bandwidth at matched memory budgets.

The setting is representative of compact UMA devices. For example, the 32 GB ROG Flow Z13 couples a Ryzen AI Max+ 395 and Radeon 8060S to one 32 GB LPDDR5X pool (ASUS, 2025); the OS, applications, KV cache, and model state leave only a fraction of that memory for experts. Caching remains valuable, but it cannot eliminate misses under a constrained expert budget.

Figure 3(a) illustrates the effect for DeepSeek-V2-Lite. At 16 GB, ZipMoE reaches 3.80 tokens/s; at 12 GB it reaches 1.96 tokens/s (a 48% reduction). Our pipeline changes from 7.94 to 5.79 tokens/s (27%), giving 2.09× and 2.95× matched-budget speedups. Figure 5 shows the same budget dependence for Qwen1.5 and DeepSeek.

Serial no-pipeline cache baseline. Table 18 reports the cold-miss and 8 GB serial, no-pipeline baseline; the 12 GB values appear beside the schedule results in Tables 2 and 3. This makes the effect of cache residency explicit before the concurrent, readiness-driven schedule is applied: a hit avoids an SSD fetch, while a miss remains exposed to the one-expert-at-a-time critical path. The 8 GB rows retain their measured hit rates; "cold miss" disables reuse entirely. At 12 GB the serial hit rates are 50.1%, 62.1%, 52.8%, and 57.7% for Qwen1.5, DeepSeek, Gemma4, and Qwen3.5, respectively. The corresponding throughput standard deviations are 0.102, 0.182, 0.083, and 0.010 tokens/s; their 95% confidence intervals are [3.771, 4.281], [2.865, 3.768], [2.328, 2.743], and [2.452, 2.499] tokens/s.

The serial baseline improves with cache residency, but its queue drains at each expert boundary. It therefore isolates a different effect from the full pipeline: the table quantifies what caching contributes before inter-expert concurrency, chunk-level readiness, and operator-aware overlap are enabled. The 12 GB full/serial ratio in Table 2 is a ratio of reported means, without a paired confidence interval.

Sensitivity to replacement policy. Table 19 holds the Qwen1.5 generation workload and execution schedule fixed while varying the decoded-expert cache policy and slot capacity. We run the

Table 18: GB10 serial no-pipeline baseline: cold miss and 8 GB cache.

Model	Cache regime	Tok/s mean ± sd	95% CI (tok/s)	TTFT (s)	TBT (s)	Hit rate
Qwen1.5-MoE-A2.7B	Cold miss	2.285 ± 0.034	[2.201, 2.369]	4.574	0.3720	–
DeepSeek-V2-Lite	Cold miss	1.623 ± 0.101	[1.373, 1.874]	5.439	0.5411	–
Gemma4-26B-A4B	Cold miss	1.309 ± 0.076	[1.121, 1.496]	5.795	0.6859	–
Qwen3.5-35B-A3B	Cold miss	1.489 ± 0.071	[1.312, 1.666]	9.885	0.5264	–
Qwen1.5-MoE-A2.7B	8 GB	3.273 ± 0.103	[3.016, 3.529]	4.255	0.2431	36.8%
DeepSeek-V2-Lite	8 GB	2.618 ± 0.114	[2.335, 2.901]	5.209	0.3058	48.6%
Gemma4-26B-A4B	8 GB	1.437 ± 0.133	[1.107, 1.768]	5.798	0.6190	9.2%
Qwen3.5-35B-A3B	8 GB	2.153 ± 0.014	[2.118, 2.188]	9.852	0.3155	43.1%

Table 19: GB10 Qwen1.5 end-to-end cache-policy ablation (two runs per entry).

Memory setting		LRU			LFU			Hybrid-0.5		
Budget (GB)	Slots	Tok/s	Hit (%)	TBT (s)	Tok/s	Hit (%)	TBT (s)	Tok/s	Hit (%)	TBT (s)
5.01	4	4.489	19.8	0.1808	4.500	19.8	0.1802	4.498	19.8	0.1805
6.56	8	4.867	26.8	0.1661	4.922	28.0	0.1638	4.748	24.3	0.1711
9.65	16	5.487	37.5	0.1463	5.728	41.4	0.1385	5.523	38.4	0.1438
15.84	32	7.066	57.1	0.1111	7.843	63.7	0.1010	7.262	59.2	0.1121
22.03	48	12.790	84.7	0.0587	16.797	92.1	0.0439	12.055	83.2	0.0670

end-to-end generation loop on ShareGPT prompts with 64 generated tokens and eight I/O workers; each entry averages two runs. The budget column gives the reported memory budget, with the corresponding number of cache slots shown separately. LFU uses a recency-decayed frequency score to rank experts. Hybrid-0.5 is a segmented LRU cache with half of its slots in the protected segment.

At four slots, all three policies have the same 19.8% hit rate and nearly identical throughput: there is too little capacity to exploit long-term expert reuse. As capacity rises, LFU retains more frequently accessed experts and reduces the remaining SSD work. Its advantage over LRU grows from 1.1% at eight slots to 31.3% at 48 slots; the latter coincides with a hit-rate increase from 84.7% to 92.1%. Hybrid-0.5 does not improve consistently over LRU in this workload. These policy comparisons use only two runs per cell and establish sensitivity under this routing trace rather than a universal ranking of replacement algorithms. All policies retain cache misses, which still use the same concurrent streaming path.