RETHINKING EXPERT PREFETCHING FOR OFFLOADED DIFFUSION MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Diffusion mixture-of-experts (MoE) language models repeatedly process the same positions across denoising steps, making expert routing highly stable over time. This makes short-horizon expert prefetching a natural way to hide host-to-device transfer latency. We find that the same stability also limits what prefetching adds beyond a recency-based cache: when the GPU cache can retain most of one denoising step's expert working set, many experts predicted for the next step are already resident under LRU. Motivated by this observation, we separate predictive cache placement from transfer-latency hiding and use a demand-driven asynchronous offloading design: expert residency follows observed demand through a per-layer LRU cache, while cache misses are transferred asynchronously and overlapped with computation, with no speculative prefetching. This isolates the additional value of prediction beyond cache recency and asynchronous demand loading. Deterministic miss accounting shows that, near and above the working-set scale, last-step prefetching reduces few or no demand misses beyond LRU while still introducing speculative transfers; in contrast, an oracle with true future routing cuts misses by $6.7\times$. Bandwidth throttling further shows that these speculative transfers compete with demand traffic for host-to-device bandwidth. Across LLaDA-2.0-mini and the $6\times$ larger LLaDA-2.0-flash on an A5000 and an H200, our demand-driven asynchronous design improves throughput by 4.9–11.8% over asynchronous last-step prefetching near and above the per-step working-set scale, with all paired 95% confidence intervals excluding zero. The same ordering holds across five tasks and batch sizes 1–8. The effect depends on cache capacity: the no-prefetch advantage is significant for $cap/W = 0.83$–1.73, but falls to $+0.73\%$ (95% CI $[-1.16, +2.63]$) at $cap/W = 0.43$, where the cache can no longer retain the recent working set. An interval-refresh placement policy inspired by TIDE does not reduce misses below LRU in our execution setting. Head-to-head against TIDE's released implementation at the same expert budget on the same GPU, our system achieves a $14.4\times$ speedup ($6.1\times$ at half the budget). These results show that expert prefetching in offloaded diffusion MoEs should be evaluated by the additional misses it avoids beyond cache residency and asynchronous miss handling, not by routing predictability alone.

1 INTRODUCTION

Mixture-of-experts (MoE) models activate only a small subset of parameters for each token, making expert offloading a practical way to serve models whose full expert bank does not fit in GPU memory (Fedus et al., 2022; Lepikhin et al., 2021; Jiang et al., 2024). In an offloaded system, experts reside in host memory and a smaller GPU cache holds the experts needed by the current forward pass. A routed expert that is not resident must be transferred over the CPU–GPU interconnect before its computation can proceed, making expert movement a central source of inference overhead.

Diffusion language models introduce a distinctive temporal structure to this problem. Rather than generating one token at a time, they repeatedly refine a block of tokens over multiple denoising steps, running the model again over largely the same positions at each step (Austin et al., 2021; Lou et al., 2024; Nie et al., 2025). As a result, expert routing is highly stable across adjacent denoising steps. This naturally motivates predictive placement: routing at step t provides a strong signal for which experts will be needed at step $t{+}1$, so those experts can be transferred before they are

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.

demanded. Recent systems for offloaded MoE inference, including work on diffusion MoEs, exploit this temporal structure through expert prediction or periodic placement updates (Eliseev & Mazur, 2023; Xue et al., 2024; Hwang et al., 2024; Chen et al., 2026).

Prediction, however, is not the only mechanism that can exploit temporal routing stability. If step $t+1$ uses many of the same experts as step t, then an expert cache that retains recently used experts may already contain much of the next-step working set. The remaining misses can be loaded asynchronously, overlapping their transfers with available computation. A future expert may therefore be easy to predict while requiring no proactive movement because it is already resident. The relevant question is not only how well future routing can be predicted, but how much explicit prediction adds beyond cache recency and asynchronous demand loading.

With asynchronous demand loading, prefetching does not improve performance when the GPU cache is large enough to retain most of one denoising step's expert working set. Across LLaDA-2.0-mini and LLaDA-2.0-flash (Bie et al., 2025) on an A5000 and an H200, turning off prefetching improves throughput by 4.9–11.8% over last-step prefetching, with every paired 95% confidence interval excluding zero. Relative to a synchronous last-step baseline, no-prefetch plus asynchronous demand loading improves throughput by 18.3% in paired eager measurements and by 20.7% in a compiled run. The same ordering holds across five tasks and batch sizes 1–8, with no measured loss in accuracy. Compared with TIDE (Chen et al., 2026), the closest published system for offloaded diffusion MoE inference, we obtain $14.4\times$ and $6.1\times$ speedups at expert budgets of 128 and 64 per layer on the same GPU.

The reason comes from the same routing stability that motivates prefetching in the first place. Consecutive denoising steps reuse many of the same experts, so a last-step predictor often selects experts that are already resident under LRU. When the cache can hold the per-step working set, last-step prefetching reduces few or no demand misses beyond plain cache recency. An oracle that knows the next step's routing can still reduce misses substantially, showing that better placement is possible in principle. The issue is that practical short-horizon prediction adds little over the cache state already produced by the previous step. Its speculative transfers still use the same host-to-device link as demand transfers, and therefore add communication when they do not avoid enough misses to compensate for it.

Our contributions are:

- **Insight and characterization.** We show that high cross-step routing predictability does not necessarily make expert prefetching useful in offloaded diffusion MoEs. When recent expert demand can remain resident in an LRU cache, many experts predicted for the next denoising step are already available, limiting the incremental benefit of predictive placement. We further characterize this effect by cache capacity relative to the per-step working set, cap/W.
- **System design and analysis.** We develop a demand-driven asynchronous offloading design that separates cache placement from latency hiding: expert residency follows observed demand through LRU, while cache misses are transferred asynchronously and overlapped with computation. Deterministic miss accounting, an oracle analysis, and bandwidth throttling explain when and why speculative prefetching helps or hurts.
- **Evaluation.** Across LLaDA-2.0-mini and LLaDA-2.0-flash on A5000 and H200 GPUs, our no-prefetch asynchronous design improves throughput by 4.9–11.8% over asynchronous last-step prefetching near and above the per-step working-set scale, with the same ordering across five tasks and batch sizes 1–8. Compared with TIDE's released implementation on the same A5000 and expert budget, it achieves a $14.4\times$ speedup at $cap = 128$ and $6.1\times$ at $cap = 64$.

2 Background and Motivation

2.1 Offloaded Diffusion MoE Inference

MoE models increase parameter capacity while activating only a subset of experts per token. This design follows sparsely gated MoEs and subsequent routing and scaling work (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022; Lewis et al., 2021; Du et al., 2022; Zoph et al., 2022; Zhou et al., 2022; Rajbhandari et al., 2022), and is used by large open models including Mixtral, DeepSeekMoE, DeepSeek-V2, and OLMoE (Jiang et al., 2024; Dai et al., 2024; DeepSeek-AI, 2024;

Muennighoff et al., 2025). When the expert bank exceeds GPU memory, inference systems keep only part of it on device and fetch or execute missing experts on demand. Existing systems therefore optimize expert caching, placement, prefetching, and CPU–GPU orchestration across single-GPU and distributed settings (Eliseev & Mazur, 2023; Yi et al., 2025; Du et al., 2024; Xue et al., 2024; Hwang et al., 2024; Zhong et al., 2024; Song et al., 2024; Tang et al., 2024; Cao et al., 2025; Cai et al., 2024; Kamahori et al., 2025; He et al., 2026; Yu et al., 2026b; Zhong et al., 2025; Doucet et al., 2026; Zhu et al., 2025a). A recurring approach is to predict future expert demand so that transfer can be hidden or avoided; the benefit of doing so depends on how much of that demand is not already present in the cache (Liang et al., 2026; Lin et al., 2025).

Diffusion language models change the temporal access pattern of this offloading problem. They generate by iterative refinement rather than strict left-to-right decoding (Austin et al., 2021; Li et al., 2022; Lou et al., 2024; Sahoo et al., 2024; Arriola et al., 2025). Recent work scales this paradigm to general-purpose and MoE language models, including LLaDA and LLaDA2.0 (Nie et al., 2025; Bie et al., 2025), while inference systems exploit reuse across denoising steps (Wu et al., 2026; Liu et al., 2026; Ma et al., 2025). In LLaDA2.0, a block of 32 masked tokens is refined over multiple denoising steps, each requiring another forward pass through the MoE layers. The same positions are therefore revisited repeatedly, creating strong temporal locality in expert routing.

2.2 Temporal Routing Locality and Expert Prefetching

The routing locality is strong enough to make adjacent-step prediction a reasonable design choice. On our GSM8K traces, tokens with router-input similarity $\texttt{cos_prev}> 0.999$ retain almost the same top-8 expert set across adjacent steps (Jaccard 0.98), while tokens with $\texttt{cos_prev}< 0.90$ have overlap 0.14; across tokens, the correlation between hidden-state similarity and expert-set overlap is $r = 0.875$. TIDE exploits this cross-step stability with interval-based refresh (Chen et al., 2026), and related systems use trace-based caches, learned or lookahead predictors, earlier routing signals, and spatio-temporal structure to stage experts before demand (Xue et al., 2024; Song et al., 2024; He et al., 2026; Yu et al., 2026b;a; Zhu et al., 2025b; Wang et al., 2025; Zhao et al., 2026). If routing at step t closely predicts routing at step $t{+}1$, prefetching appears to offer a direct way to hide transfer latency.

The same stability also means that step t and step $t{+}1$ often need the same experts. An LRU cache passively retains experts used recently, so many of the experts selected by a last-step predictor may already be resident. This is the key distinction for our setting: prediction accuracy measures whether the future expert set can be inferred, while system benefit depends on whether that prediction changes the cache state enough to avoid demand misses.

2.3 Cache Recency, Asynchronous Loading, and Working Set

Recency and prediction are therefore two ways to exploit the same temporal structure. Recency keeps experts from the previous step in the GPU cache; prefetching proactively moves experts that a predictor expects to need next. Any remaining miss can also be served asynchronously: the required H2D copy is issued on a side stream and synchronized only before the routed expert computation, allowing part of the transfer to overlap with other work. Modern offloading systems likewise overlap expert movement with computation through asynchronous or pipelined I/O (Cao et al., 2025; Yu et al., 2026a; Doucet et al., 2026). The relevant comparison is therefore not prediction against a fully synchronous cache, but prediction against a cache that already exploits both recency and asynchronous miss handling.

We use the per-step expert working set W to describe how much recent routing demand the cache can retain. W is the mean number of distinct experts that one MoE layer routes to in a forward pass, following the classical working-set view of memory locality and LRU-style replacement (Denning, 1968; Mattson et al., 1970). Measured from our cache statistics, $W \approx 77$ for LLaDA-2.0-mini and $W \approx 74$ for LLaDA-2.0-flash. With per-layer cache capacity cap, the ratio cap/W indicates whether the recent expert set can largely remain resident. When $cap \gtrsim W$, recency can preserve much of the predictable next-step demand; when $cap \ll W$, capacity misses leave more room for placement policy to matter.

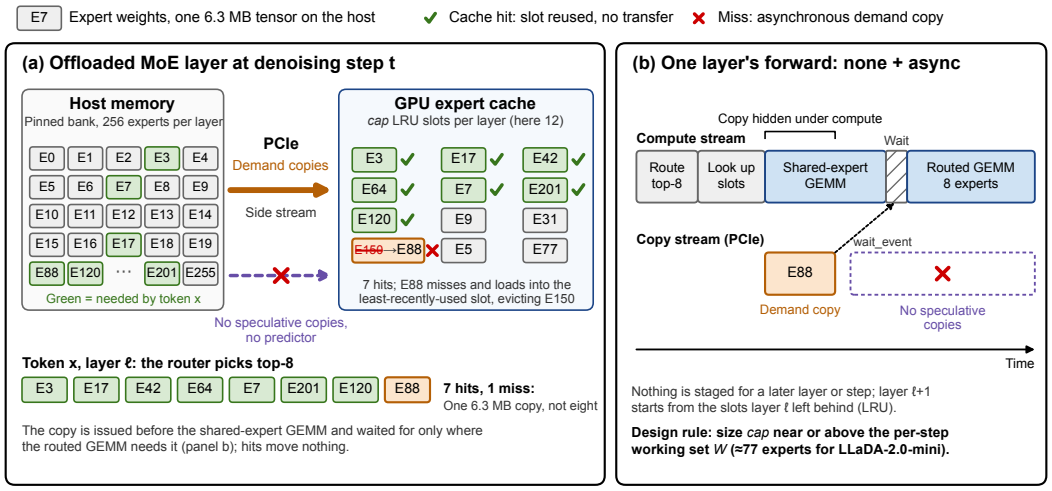


Figure 1: **Demand-driven asynchronous expert offloading.** **(a)** What moves. Experts live in pinned host memory, each MoE layer owns a GPU cache of *cap* LRU slots, and the only host-to-device traffic is demand copies of missing experts on a side stream; there is no predictor and no prefetch stream. In the example, seven of a token's eight routed experts hit and E88 loads into the least-recently-used slot. **(b)** When it moves. Route, look up slots, issue the copy for the miss, run the shared-expert GEMM while it lands, wait only for that copy, then run the routed GEMM. Nothing is staged for a later layer or step. Expert IDs and the 12-slot cache are illustrative.

3 Demand-Driven Asynchronous Expert Offloading

The discussion in §2 suggests separating two roles that are often coupled in expert prefetching: deciding which experts should remain resident, and hiding the latency of experts that are not resident. A prefetch-based system uses predicted future routing for both. It changes the cache state before demand arrives and relies on the resulting early transfers to reduce or hide future miss latency. In diffusion MoEs, however, temporal locality already causes many next-step experts to remain in the cache after the current denoising step. Speculative movement is useful only when it improves that cache state beyond what observed demand has already produced.

Our design therefore uses *demand-driven* expert movement. Cache residency is determined by actual routed accesses through LRU replacement, while misses are transferred asynchronously when they occur. No expert is moved solely because it is predicted to be needed in a future denoising step. Recency captures reuse and asynchronous demand loading hides part of the remaining transfer latency, the two properties prefetching is usually meant to provide. The design is deliberately simple; its purpose is to remove speculative movement without giving up either caching or overlap. Figure 1 summarizes it.

3.1 Separating Residency from Latency Hiding

Consider two consecutive denoising steps, t and $t{+}1$. If their routing sets overlap strongly, a predictor can infer many experts needed at $t{+}1$ from step t, but those same experts were just accessed at step t, so a recency-based cache has a good chance of retaining them without any prediction. Prefetching changes system behavior only for experts that are both predicted and absent from the cache state that recent demand would leave behind: a high routing overlap can make a predictor accurate while leaving little useful movement to perform, and only when recent demand cannot stay resident, because the cache is much smaller than the working set, does prediction have room to alter placement. Latency hiding is a separate matter. A miss need not block all useful work as soon as it is discovered; the missing expert can be copied on another CUDA stream and synchronized only when the routed computation needs it, which overlaps the copy with the shared-expert GEMM without predicting anything. Our design therefore uses prediction only where it adds value beyond these two mechanisms, rather than as the default way to obtain overlap.

3.2 Recency-Based Residency and Asynchronous Misses

Each MoE layer maintains an independent GPU cache of *cap* expert slots, while the full expert bank remains in pinned host memory. The routed grouped-GEMM reads expert weights directly from these slots. If a routed expert is already resident, execution proceeds without expert movement. On a miss, the cache selects a victim using LRU replacement and loads the requested expert from host memory.

LRU records the routing history implicitly through residency: experts accessed recently remain in the cache until capacity pressure evicts them, which matches the temporal structure of §2.2. Our no-prefetch policy, denoted `none`, performs no speculative cache mutation; experts enter the cache only after observed demand.

A demand miss is issued on a side CUDA stream. We insert a `wait_event` only immediately before the routed GEMM requires the copied weights, allowing the transfer to overlap with the shared-expert GEMM and other available computation. We refer to this as *asynchronous demand loading*. Under the corresponding synchronous mode, execution blocks while the missing expert is transferred.

The resulting `none`+async path is fully demand-driven: a hit incurs no transfer, a miss launches only the copy the observed routing requires, and execution waits only when that expert is needed. No separate stream of speculative copies changes the cache ahead of demand.

3.3 Temporal Prefetch Baseline and Design Space

To isolate the value of predictive movement, we compare the demand-driven path with a last-step temporal prefetcher, denoted `last_step`. For each layer, `last_step`uses the previous denoising step's routing as its prediction of the next step. Experts are ranked by routing weight, and predicted experts that are not already resident are staged into the cache. Given the adjacent-step routing stability in §2.2, this is a natural short-horizon predictor for diffusion MoE inference rather than an intentionally weak baseline.

The prefetch pipeline runs one layer ahead. After layer K completes its routed GEMM, speculative copies for layer $K{+}1$ are issued on a dedicated prefetch stream. Before layer $K{+}1$ handles its demand misses, outstanding prefetch copies must complete so that their destination slots can be committed. Prefetching can therefore help when these early transfers prevent future demand misses, but it can also add H2D traffic or replace residents that would otherwise have remained cached.

We independently vary temporal prefetching and demand-miss handling, producing four configurations:

`last_step` + sync, `last_step` + async, `none` + sync, `none` + async.

This 2×2 design separates two effects that are otherwise easy to conflate. Comparing synchronous and asynchronous loading measures the benefit of hiding demand-transfer latency. Comparing `last_step`and `none`under the same miss-loading mode measures the additional value of predictive placement. Our main comparison is therefore `last_step`+async versus `none`+async: both can overlap unavoidable demand transfers with computation, while only the former performs speculative expert movement. Figure 2a illustrates these two execution paths.

4 Evaluation

4.1 Experimental Setup

We evaluate LLaDA-2.0-mini and LLaDA-2.0-flash (Bie et al., 2025) with a modified dInfer engine (Ma et al., 2025). Mini has 19 MoE layers with 256 experts per layer and top-8 routing; flash has 31 MoE layers with the same routing structure and $4\times$ larger experts. Mini runs primarily on a 24 GB A5000 ($\approx$17 GB/s measured H2D bandwidth); flash and a mini replication run on a 141 GB H200. Unless noted otherwise, experiments use bf16 weights, 30 held-out GSM8K prompts, 256 generated tokens, and $cap = 128$. Throughput is generated tokens per second. Main eager comparisons use eight interleaved paired rounds with 95% confidence intervals; their decode trajectories are identical, so miss and transfer counts are deterministic.

Table 1: **Crossing placement and miss-loading policy** (LLaDA-2.0-mini, A5000, GSM8K, $cap = 128$). Eager values are mean tps $\pm$ 95% CI over 8 interleaved rounds on the same decode trajectory (2,237 forwards); paired comparisons are reported in the text. Compiled values are single deployed-mode runs on an earlier engine revision; percentages are relative to `last_step`+sync.

prefetch	miss loading	eager tps (mean $\pm$ 95% CI)	compiled tps (single run)
`last_step`	sync	39.56 ± 0.14	30.22
`last_step`	async	44.39 ± 1.57	31.97 (+5.8%)
`none`	sync	41.06 ± 2.27	32.77 (+8.4%)
`none`	async	$\mathbf{46.78 \pm 1.63}$	**36.48** (+**20.7**%)

Table 2: **Paired advantage of `none`+async over `last_step`+async across cache budgets, models, and GPUs** (GSM8K, eager, bf16; mean tps $\pm$ 95% CI over $n = 8$ interleaved rounds unless noted). W is the measured per-step working set (77 mini, 74 flash). The final row is a statistical null. Flash on a 24 GB A5000 with quantized experts is reported in Appendix A.8.

GPU	model	*cap*	cap/W	`none`+async	`last_step`+async	paired advantage (95% CI)
A5000	mini	128	1.66	46.78 ± 1.63	44.39 ± 1.57	$+5.55\%$ $[+0.25, +10.85]$
A5000	mini	64	0.83	15.28 ± 0.06	14.42 ± 0.02	$+5.99\%$ $[+5.56, +6.42]$
H200	mini	128	1.66	69.00 ± 6.39	62.09 ± 6.38	$+11.83\%$ $[+1.01, +22.65]$
H200	mini	64	0.83	32.14 ± 1.05	30.65 ± 0.33	$+4.86\%$ $[+1.84, +7.87]$
H200	flash	128	1.73	28.71 ± 0.81	26.74 ± 0.13	$+7.38\%$ $[+4.18, +10.57]$
H200	flash	64	0.86	8.39 ± 0.17	7.58 ± 0.10	$+10.70\%$ $[+8.87, +12.54]$
H200	flash	32	0.43	4.13 ± 0.09	4.10 ± 0.06	$+0.73\%$ $[-1.16, +2.63]$ n.s. ($n = 6$)

4.2 Main Throughput Results

Table 1 separates predictive placement from miss handling. Async loading improves eager throughput by $+14.4\%$ without prefetching and $+12.2\%$ with `last_step`. Once async is enabled, removing `last_step` adds another $+5.55\%$ (95% CI $[+0.25, +10.85]$). Relative to `last_step`+sync, `none`+async improves throughput by $+18.3\%$ paired (CI $[+14.1, +22.5]$) and by $+20.7\%$ in the compiled run. Under synchronous loading alone, removing prefetch is only a $+3.8\%$ difference with a confidence interval spanning zero. Thus, the main change is not simply "prefetch off": asynchronous demand loading changes the value of the speculative work.

Across models, GPUs, and cache sizes near the working-set scale, the same ordering holds. Table 2 shows a significant `none`+async advantage of $+4.86$–11.83% for $cap/W = 0.83$–1.73. The only deep-small-cache point, $cap/W = 0.43$, is a null ($+0.73\%$, 95% CI $[-1.16, +2.63]$); we analyze this regime in §4.4.

4.3 Why Prefetching Adds Little Once Misses Are Asynchronous

Cache benefit. Table 3 separates routing predictability from useful cache movement. At $cap = 128$, `last_step` issues 17,882 speculative transfers yet slightly increases demand misses (211,683 vs. 209,016). An oracle with true next-step routing cuts misses to 31,290, a $6.7\times$ reduction, so future information can help; the practical last-step predictor simply adds little beyond the cache state already left by recent demand. At $cap = 64$, `last_step` gains only about one hit-rate point for 204k speculative transfers, whereas the oracle gains roughly nine. Additional predictor variants are reported in Appendix A.3.

Transfer cost. Expert movement remains material even at high hit rate: at $cap = 128$, one warm forward contains about 38 ms of compute and 26 ms of expert DMA; asynchronous loading hides about 9 ms of that DMA (Figure 3b). Throttling H2D bandwidth while keeping miss and move counts fixed makes the cost of prefetching grow: at $cap = 64$, the `none`-over-`last_step` advantage rises from $+8.3\%$ at 16.8 GB/s to $+17.2\%$ at 3.8 GB/s. At $cap = 128$, the extra time

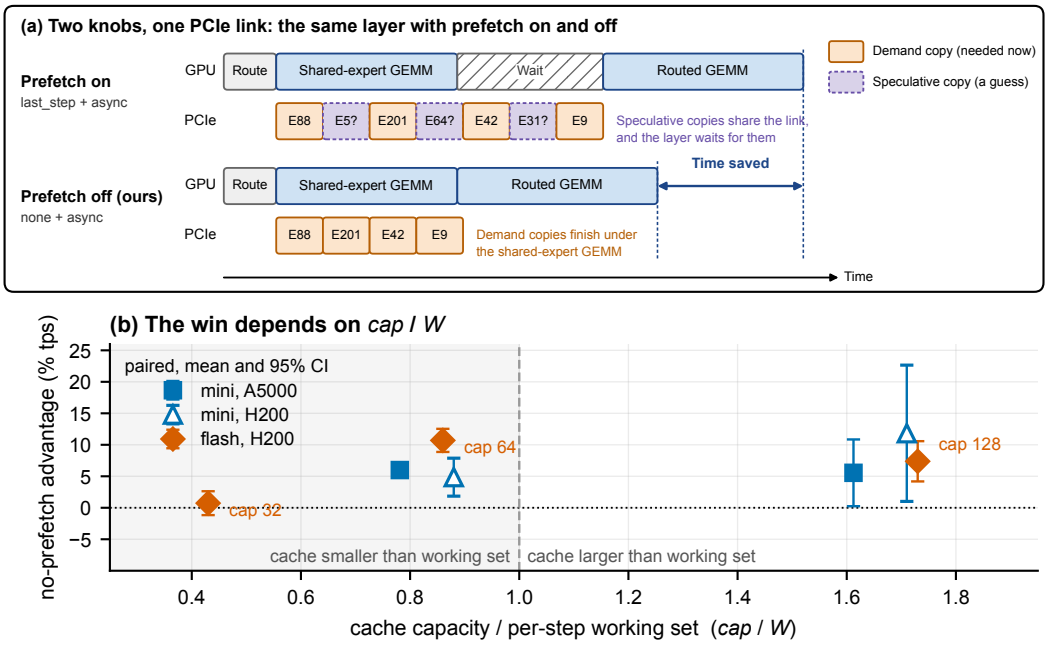


Figure 2: **Asynchronous demand loading changes the value of temporal prefetching, and the effect depends on cache capacity relative to the working set.** **(a)** One MoE layer at one denoising step with prefetch on and off. Demand copies of missing routed experts (orange) run on a side stream under the shared-expert GEMM. Last-step prefetch adds speculative copies (purple) for the next layer, which share the PCIe link and are waited for before the routed GEMM. Expert IDs are illustrative. **(b)** Paired throughput advantage of `none`+async over `last_step`+async against cap/W (95% CIs over 8 interleaved rounds; 6 for flash $cap = 32$). The advantage is significant near and above the working-set scale and becomes a statistical null deep below it.

Table 3: **Predictability and cache benefit are different.** GSM8K, eager, mini, A5000. Last-step prefetch does not reduce demand misses below LRU above the working set and buys about one point of hit rate just below it. An oracle with true future routing reveals substantially larger predictive headroom, at the cost of additional transfers.

cap	prefetch	hit rate	demand misses	speculative transfers
128	`none`	93.61%	209,016	0
128	`last_step`	93.53%	211,683	17,882
128	oracle	99.04%	31,290	177,792
64	`none`	70.87%	952,480	0
64	`last_step`	71.93%	917,680	204,320
64	oracle	79.75%	662,130	432,493

from prefetching rises from 1.3 to 10.1 ms/token as the link is slowed 8.8×. The throttle does not separate link contention from waiting on outstanding prefetch copies; both are costs of speculative movement in the evaluated path. Together with Table 3, this explains the throughput result: `last_step` changes the cache little in this regime while still consuming transfer bandwidth.

4.4 Cache Regime and Robustness

The mechanism predicts a cache-capacity boundary. Near the working-set scale, LRU retains much of the next-step expert set while transfer traffic remains large enough for speculative copies to matter. Far below it, recent demand no longer fits and placement has more opportunity to recover capacity misses. Figure 2b matches this picture: the `none`+async advantage is significant for $cap/W =$

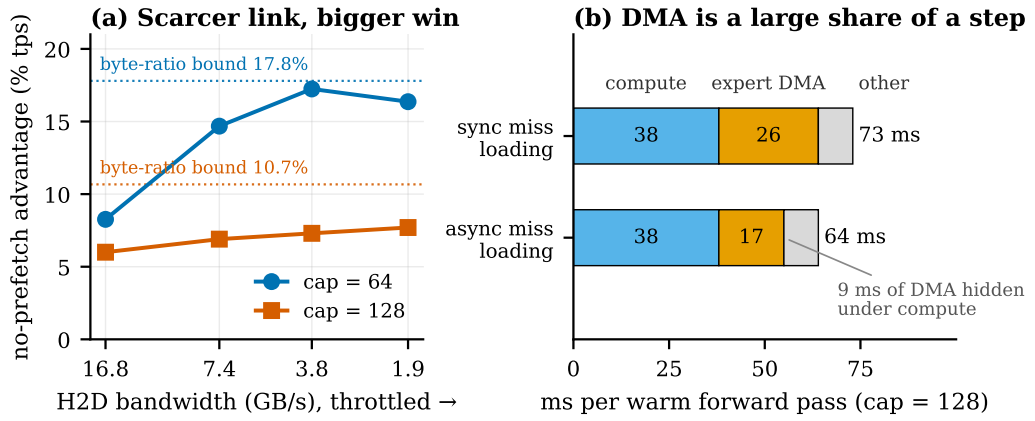


Figure 3: **Speculative movement consumes a transfer resource that remains important under asynchronous loading.** **(a)** Throttling H2D bandwidth increases the throughput advantage of `none` over `last_step`, while miss and move counts remain fixed. Dotted lines are the transfer-byte-ratio bounds. **(b)** Latency budget of one warm forward at $cap = 128$ (mini, A5000): 38 ms compute, 26 ms expert DMA under synchronous loading, and 9 ms of that DMA hidden by asynchronous loading.

0.83–1.73, but falls to $+0.73\%$ (95% CI $[-1.16, +2.63]$) at $cap/W = 0.43$. There, `last_step` raises hit rate from 38.4% to 41.0% by reclaiming 121k misses with 354k speculative transfers. This is a null, not evidence that prefetching wins; it shows that the overlap between recency and short-horizon prediction weakens when the cache is far smaller than the working set. Appendix A.6 and Appendix A.8 provide additional capacity sweeps.

Batch size and tasks. Batching increases the working set: from batch 1 to 8, aggregate miss rate rises from 6.5% to 10.0%, yet `none`+async remains $+6.5$–12.8% faster in the single-run sweep (Figure 4a). Across five tasks in compiled single runs, `none`+async wins on every one: HumanEval $+43.5\%$, MBPP $+15.2\%$, MMLU $+14.9\%$, GSM8K $+14.1\%$, and IFEval $+13.5\%$; these are robustness checks, not paired estimates.

Alternative temporal placement. We also evaluate an interval-refresh policy inspired by TIDE (Chen et al., 2026), while keeping LRU and asynchronous demand loading fixed. Across $\tau \in \{1, 2, 4, 8, 16\}$, demand-miss rate stays near 0.064 and no tested interval exceeds `none` in the single-run throughput sweep: TIDE's default τ=2 is 5.2% slower and the best interval, τ=4, is 1.3% slower (Figure 4b; full values in Appendix A.1).

Compared with TIDE's released system. We also ran TIDE's released implementation (Chen et al., 2026) unmodified on the same A5000, model, prompts, and decoding settings at its default refresh interval (τ=2): a per-layer budget of expert modules on the GPU, refreshed every τ steps from the current step's routing counts with synchronous copies, and CPU compute for tokens routed to non-resident experts. Table 4 reports interleaved rounds against `none`+async at the same budget: we obtain a $14.4\times$ speedup with a 128-expert budget (47.2 vs. 3.3 tokens/s) and $6.1\times$ with a 64-expert budget (16.4 vs. 2.7). A longer refresh interval helps TIDE only slightly (3.4 tokens/s at τ=4, 3.5 at τ=8), and under the same answer-extraction rule it scores 22–24 of 30 GSM8K prompts against our 25. Most of this gap is engine implementation rather than placement policy: TIDE's code loops over experts in Python, moves whole modules synchronously, and spends 17% of its wall time moving experts and 19% computing on the CPU, whereas our engine runs one fused grouped GEMM per layer. The H200 rows show the same gap without offloading: with all 256 experts resident our engine is $8.3\times$ faster than TIDE's, and with a 128-expert cache it keeps 81% of its own ceiling, whereas TIDE's budget-128 run did not finish 30 prompts within our two-hour cap per run because its CPU expert path is slow on that host. Read with the predictor port above: TIDE's

Table 4: **Head-to-head against TIDE's released implementation** (LLaDA-2.0-mini, GSM8K, same prompts and decoding settings, interleaved rounds; tps mean (range) over rounds; TIDE at τ=2; speedup is the mean per-round ratio of ours to TIDE). Moves/fwd and CPU share are TIDE's expert moves per forward pass and its fraction of token-expert assignments computed on the CPU; "all" means every expert resident. †Did not finish 30 prompts within two hours per run; rate over completed prompts.

GPU	budget / *cap*	TIDE tps	none+async tps (ours)	speedup	moves/fwd	CPU share
A5000	128	3.26 (3.26–3.27)	47.15 (47.09–47.21)	14.4×	66	4.6%
A5000	64	2.71 (2.69–2.74)	16.44 (16.43–16.44)	6.1×	56	22.4%
H200	256 (all)	10.14 (10.14–10.15)	84.04 (83.11–84.97)	8.3×	0	0%
H200	128	≈ 0.9†	67.99 (67.60–68.38)	–	–	–

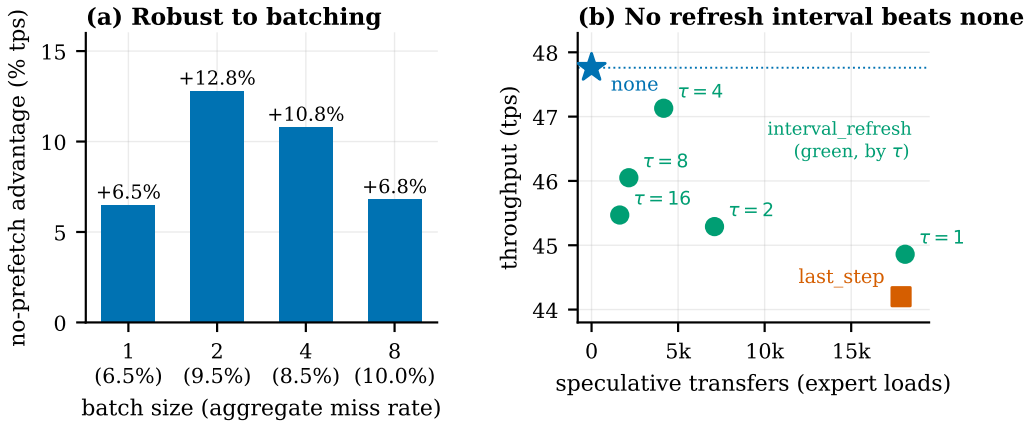


Figure 4: **Working-set stress and alternative temporal placement. (a)** The none+async advantage remains positive across batch sizes 1–8 while the aggregate miss rate increases with batch size (mini, eager, single runs). **(b)** Interval-refresh placement trades speculative transfer count against the refresh interval τ, but no interval improves on none in the single-run throughput sweep.

policy does not beat recency inside one engine, and its released implementation does not close the gap.

Accuracy. Removing speculative prefetching does not produce a measured accuracy loss. The 30-prompt GSM8K set is identical at 76.7% exact match; on 200 held-out GSM8K prompts the difference is not significant (McNemar $\chi^2 = 2.77$); and HumanEval eager outputs are bit-identical at 59.8% pass@1.

5 Conclusion

Diffusion MoE routing is highly predictable across denoising steps, but predictability alone does not make expert prefetching worthwhile. The same stability that makes adjacent-step prediction accurate also keeps many next-step experts resident under LRU, so near or above the per-step working set a short-horizon prefetcher avoids few additional misses while its speculative copies still consume the link that asynchronous demand loading needs. Across the models, GPUs, tasks, and batch sizes we evaluate, none+async is therefore faster than temporal prefetching in that regime, and no worse far below it. The design principle is simple: judge expert prefetching by the misses it avoids beyond cache residency, relative to the movement it adds, not by routing predictability alone. Whether the same holds for autoregressive MoE offload with asynchronous demand loading is the natural next question.

AI USE STATEMENT

In this work, we used generative AI tools for polish writing and research execution such as code implementation. We have not used generative AI tools for idea initiation and retrieval or discovery (e.g., finding related work), and usage of generative AI for dataset generation and formal mathematical proof is not applicable to this work. We have reviewed all AI-assisted work, including writing recommendation, code implementation, and experimental result collection. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This paper presents theoretical and methodological contributions and does not involve human subjects, data collection, or the release of new datasets or models. We do not foresee any ethical concerns arising from this work.

REPRODUCIBILITY STATEMENT

All throughput and miss/move counts come from a modified dInfer engine; the expert-cache module, the `interval_refresh` policy, and the benchmark harness and evaluation protocol are described in §4.1. Miss and prefetch-move counts are deterministic given a decode trajectory and are the basis of our mechanism claims. Prompt sets are fixed held-out splits. A5000 experiments ran on shared cluster nodes; H200 experiments ran on a single cloud (RunPod) H200 SXM instance under a 234 GiB memory cgroup, with the same software pins as the A5000 runs (PyTorch 2.8.0+cu128, vLLM 0.10.2, transformers 5.8.1). We will release the cache module, policies, drivers, and result files, including per-round data for every paired study.

REFERENCES

Marianne Arriola, Aaron Gokaslan, Justin T. Chiu, Zhihan Yang, Zhixuan Qi, Jiaqi Han, Subham Sekhar Sahoo, and Volodymyr Kuleshov. Block diffusion: Interpolating between autoregressive and diffusion language models. In *International Conference on Learning Representations*, 2025.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993, 2021.

Tiwei Bie, Maosong Cao, Kun Chen, Lun Du, Mingliang Gong, Zhuochen Gong, Yanmei Gu, Jiaqi Hu, Zenan Huang, Zhenzhong Lan, Chengxi Li, Chongxuan Li, Jianguo Li, Zehuan Li, Huabin Liu, Lin Liu, Guoshan Lu, Xiaocheng Lu, Yuxin Ma, Jianfeng Tan, Lanning Wei, Ji-Rong Wen, Yipeng Xing, Xiaolu Zhang, Junbo Zhao, Da Zheng, Jun Zhou, Junlin Zhou, Zhanchao Zhou, Liwang Zhu, and Yihong Zhuang. Llada2.0: Scaling up diffusion language models to 100b. *arXiv preprint arXiv:2512.15745*, 2025.

Ruisi Cai, Yeonju Ro, Geon-Woo Kim, Peihao Wang, Babak Ehteshami Bejnordi, Aditya Akella, and Zhangyang Wang. Read-me: Refactorizing llms as router-decoupled mixture of experts with system co-design. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, pp. 715–730, 2025. doi: 10.1145/3669940.3707267.

Zhiben Chen, Youpeng Zhao, Yang Sui, Jun Wang, and Yuzhang Shang. Tide: Efficient and lossless moe diffusion llm inference with i/o-aware expert offload. *arXiv preprint arXiv:2605.20179*, 2026.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong

Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1280–1297, 2024. doi: 10.18653/v1/2024.acl-long.70.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

Peter J. Denning. The working set model for program behavior. *Communications of the ACM*, 11 (5):323–333, 1968. doi: 10.1145/363095.363141.

Zachary Doucet, Rishi Sharma, Martijn de Vos, Rafael Pires, Anne-Marie Kermarrec, and Oana Balmau. Harmoeny: Efficient inference of moe models. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2026. arXiv:2506.12417.

Nan Du, Yanping Huang, Andrew M. Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, Barret Zoph, Liam Fedus, Maarten P. Bosma, Zongwei Zhou, Tao Wang, Emma Wang, Kellie Webster, Marie Pellat, Kevin Robinson, Kathleen Meier-Hellstern, Toju Duke, Lucas Dixon, Kun Zhang, Quoc V. Le, Yonghui Wu, Zhifeng Chen, and Claire Cui. Glam: Efficient scaling of language models with mixture-of-experts. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 5547–5569, 2022.

Zhixu Du, Shiyu Li, Yuhao Wu, Xiangyu Jiang, Jingwei Sun, Qilin Zheng, Yongkai Wu, Ang Li, Hai Li, and Yiran Chen. Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models. In *Proceedings of Machine Learning and Systems*, volume 6, 2024.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*, 2023.

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.

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. arXiv:2410.17954.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, 2024.

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 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*, 2025.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6265–6274, 2021.

Xiang Lisa Li, John Thickstun, Ishaan Gulrajani, Percy Liang, and Tatsunori B. Hashimoto. Diffusion-lm improves controllable text generation. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Jingcong Liang, Siyuan Wang, Miren Tian, Yitong Li, Duyu Tang, and Zhongyu Wei. Not all models suit expert offloading: On local routing consistency of mixture-of-expert models. In *International Conference on Learning Representations*, 2026.

Shuning Lin, Yifan He, and Yitong Chen. In-depth analysis on caching and pre-fetching in mixture of experts offloading. *arXiv preprint arXiv:2511.05814*, 2025.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyan Wei, Shaobo Wang, Yichen Zhu, and Linfeng Zhang. dllm-cache: Accelerating diffusion large language models with adaptive caching. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2506.06295.

Aaron Lou, Chenlin Meng, and Stefano Ermon. Discrete diffusion modeling by estimating the ratios of the data distribution. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 32819–32848, 2024.

Yuxin Ma, Lun Du, Lanning Wei, Kun Chen, Qian Xu, Kangyu Wang, Guofeng Feng, Guoshan Lu, Lin Liu, Xiaojing Qi, Xinyuan Zhang, Zhen Tao, Haibo Feng, Ziyun Jiang, Ying Xu, Zenan Huang, Yihong Zhuang, Haokai Xu, Jiaqi Hu, Zhenzhong Lan, Junbo Zhao, Jianguo Li, and Da Zheng. dinfer: An efficient inference framework for diffusion language models. *arXiv preprint arXiv:2510.08666*, 2025.

Richard L. Mattson, Jan Gecsei, Donald R. Slutz, and Irving L. Traiger. Evaluation techniques for storage hierarchies. *IBM Systems Journal*, 9(2):78–117, 1970. doi: 10.1147/sj.92.0078.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. Olmoe: Open mixture-of-experts language models. In *International Conference on Learning Representations (ICLR)*, 2025. arXiv:2409.02060.

Shen Nie, Fengqi Zhu, Zebin You, Xiaolu Zhang, Jingyang Ou, Jun Hu, Jun Zhou, Yankai Lin, Ji-Rong Wen, and Chongxuan Li. Large language diffusion models. *arXiv preprint arXiv:2502.09992*, 2025.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 18332–18346, 2022.

Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, Aaron Gokaslan, Edgar Marroquin, Justin T. Chiu, Alexander Rush, and Volodymyr Kuleshov. Simple and effective masked diffusion language models. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. Promoe: Fast moe-based llm serving using proactive caching. *arXiv preprint arXiv:2410.22134*, 2024.

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

Liujianfu Wang, Yuyang Du, Yuchen Pan, Soung Chang Liew, Jiacheng Liu, and Kexin Chen. Odmoe: On-demand expert loading for cacheless edge-distributed moe inference. *arXiv preprint arXiv:2512.03927*, 2025.

Chengyue Wu, Hao Zhang, Shuchen Xue, Zhijian Liu, Shizhe Diao, Ligeng Zhu, Ping Luo, Song Han, and Enze Xie. Fast-dllm: Training-free acceleration of diffusion llm by enabling kv cache and parallel decoding. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2505.22618.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh K. Marina. Moe-infinity: Efficient moe inference on personal machines with sparsity-aware expert cache. *arXiv preprint arXiv:2401.14361*, 2024.

Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. Edgemoe: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 24(8):7059–7073, 2025. arXiv:2308.14352.

Enda Yu, Dezun Dong, Zhaoning Zhang, Zhe Bai, Weiling Yang, Haojie Wang, Dongsheng Li, Yongwei Wu, and Xiangke Liao. Layerscope: Predictive cross-layer scheduling for efficient multi-batch moe inference on legacy servers. In *Proceedings of the 40th ACM International Conference on Supercomputing*, pp. 881–892, 2026a. doi: 10.1145/3797905.3807834.

Hanfei Yu, Xingqi Cui, Hong Zhang, Hao Wang, and Hao Wang. 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*, pp. 176–191, 2026b. doi: 10.1145/3767295.3769319.

Yingnan Zhao, Razvan C. Bunescu, Ahmed Louri, Avinash Karanth, and Ke Wang. A spatio-temporal expert prefetching framework for efficient moe-based llm inference. *arXiv preprint arXiv:2606.15453*, 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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024. arXiv:2408.10284.

Shuzhang Zhong, Yanfan Sun, Ling Liang, Runsheng Wang, Ru Huang, and Meng Li. Hybrimoe: Hybrid cpu-gpu scheduling and cache management for efficient moe inference. In *ACM/IEEE Design Automation Conference (DAC)*, 2025. arXiv:2504.05897.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew Dai, Zhifeng Chen, Quoc Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A. Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, Jianzhe Xiao, Xinyi Zhang, Lingjun Liu, Haibin Lin, Li-Wen Chang, Jianxi Ye, Xiao Yu, Xuanzhe Liu, Xin Jin, and Xin Liu. Megascale-infer: Efficient mixture-of-experts model serving with disaggregated expert parallelism. In *Proceedings of the ACM SIGCOMM 2025 Conference*, pp. 592–608, 2025a. doi: 10.1145/3718958.3750506.

Shien Zhu, Samuel Bohl, Robin Oester, and Gustavo Alonso. Pre-attention expert prediction and prefetching for mixture-of-experts large language models. *arXiv preprint arXiv:2511.10676*, 2025b.

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

A Additional details

A.1 Interval-refresh policy

`IntervalRefreshPrefetch` maintains a per-layer token-count histogram over the previous step's routing. Every τ decode steps ("refresh steps") its `predict` returns the top-*cap* experts by that count for bulk staging; on intervening steps it returns the empty set. τ=1 therefore refreshes every step and recovers per-step prefetch (empirically matching `last_step`; Table 5). Eviction (LRU) and miss loading (async) are held fixed, so only the placement predictor varies. Table 5 reports the full sweep used in §4.4.

Table 5: Interval-refresh placement sweep (mini, GSM8K, $cap = 128$, eager, single run per configuration). The execution model is held fixed to LRU with asynchronous demand loading; miss rate remains nearly flat.

config	τ	tps	miss rate	speculative transfers	vs. `none`
`none`	–	**47.76**	0.0639	0	–
`interval_refresh`	4	47.13	0.0643	4,168	-1.3%
`interval_refresh`	8	46.05	0.0642	2,153	-3.6%
`interval_refresh`	16	45.47	0.0641	1,628	-4.8%
`interval_refresh`	2	45.29	0.0647	7,103	-5.2%
`interval_refresh`	1	44.86	0.0651	18,117	-6.1%
`last_step`	–	44.20	0.0647	17,882	-7.5%

A.2 Oracle prefetcher

The oracle is given each step's true routing by replaying a recorded trajectory and stages exactly the experts that step will access, providing the deterministic hit-rate ceiling in Table 3. It is not realizable online. It exists only to bound the headroom a predictor could capture.

A.3 Other predictors

Before settling on `last_step` as the baseline predictor we tried supplements on top of it in single runs with synchronous loading on an earlier engine revision (Table 6): a pinned set of the 64 hottest experts per layer, a two-phase cross-layer predictor with a 16-expert budget, and a capacity filling cross-layer plus frequency policy. Each moved more experts than `last_step` and each was slower, in proportion to the transfers it added; none raised the hit rate. These runs are not paired and are on a different revision from the main results, so we use them only to support the choice of baseline.

Table 6: Supplements to last-step prefetch (mini, GSM8K, cap=128, synchronous loading, single runs, earlier engine revision).

predictor	hit rate	tps	speculative transfers
`last_step`	93.67%	31.30	8.5k
`last_step` + pinned hot set (64)	93.61%	28.60	9.9k
two-phase cross-layer (budget 16)	92.18%	27.17	151k
`last_step` + cross-layer + frequency fill	89.47%	19.02	572k

A.4 Bandwidth throttle

Effective H2D bandwidth is limited by a software rate limit on the copy path; achieved bandwidth is verified by microbenchmark (unthrottled $\approx$16.8 GB/s). Miss and move counts are unchanged by throttling, which isolates bandwidth as the varied quantity in Figure 3a.

A.5 Latency budget

The budget in Figure 3b comes from an Nsight Systems kernel trace of `none`+async at cap=128 (mini, A5000, eager, four prompts, 334 forward passes) restricted to the steady generation region. Compute is the summed kernel time per forward (37–38 ms in two windows); expert DMA is the deterministic miss count per forward ($\approx$95) times the traced per-expert transfer time (269 μs). Profiling serializes stream concurrency, so the overlap is not read from the trace. It is inferred from the synchronous-versus-asynchronous wall-time gap (73 $\rightarrow$ 64 ms), which attributes 9 of the 26 ms of DMA to hiding under compute.

A.6 SINGLE-RUN CACHE-CAPACITY SWEEP

Figure 5 sweeps *cap* on mini in single eager runs. The advantage peaks near W (cap=64, +8.5%) and tapers when the cache thrashes regardless of policy (cap=48, +3.5%) and when it is roomy enough that misses are rare ($cap \geq 96$, +4.8 to +4.9%). The paired studies of Table 2 confirm significance at cap=64 and 128 on every model and GPU but do not resolve a consistent peak between them.

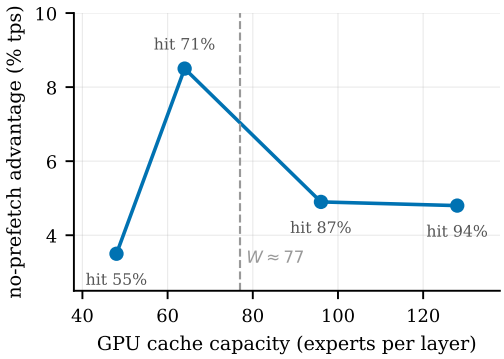


Figure 5: Single-run cache-capacity sweep (mini, A5000, eager): no-prefetch advantage (left axis) and hit rate (right axis) against cache capacity.

A.7 COMPILED MODE THRASHES BELOW THE WORKING SET

Our mechanism sweeps run in eager mode because compiled (CUDA-graph) execution degrades pathologically when $cap < W$: the cache overflows every step, its slot-map mutation breaks the captured graph, and the framework recompiles every step. This is the small-GPU offload regime, and it is a caution for deploying compiled MoE inference under tight cache budgets.

A.8 FLASH ON A 24 GB GPU

Before the H200 runs we ran flash on the A5000, where 24 GB holds only 16 bf16 flash experts per layer (6% of experts, deep thrash). To reach the regimes of interest we stored experts weight-only quantized (int8 per-channel for cap=32 and int4 group-128 for cap=64, through vLLM's `w8a16`/`w4a16` fused-MoE paths) and streamed the 206 GB checkpoint into ∼100 / 50 GB of pinned pools; on mini this preserves GSM8K accuracy (24 and 23 of 30 versus 25 in bf16 for that engine revision; bf16 itself varies between 23 and 25 of 30 across runs). Both arms of a pair share the same quantized model, so the forward count matches across arms. Table 7 gives the results, which first showed the null deep below W: significant near W, null deep below it. The margins are smaller than the bf16 H200 rows because quantized experts move fewer bytes per miss (int4 moves the same 6.3 MB as a mini expert), which shrinks the DMA share of a step.

Table 7: Flash on the 24 GB A5000 with quantized expert storage (GSM8K, eager, n=8 interleaved paired rounds; tps mean ± 95% CI).

storage	*cap*	cap/W	`none`+async	`last_step`+async	paired advantage (95% CI)
int4 g128	64	0.86	8.40 ± 0.31	8.11 ± 0.28	$+3.57\%$ $[+1.38, +5.75]$
int8	32	0.43	3.31 ± 0.06	3.25 ± 0.05	$+1.92\%$ $[-0.65, +4.49]$ n.s.