Temporal Expert Caching for Accelerating Inference in Mixture-of-Experts Diffusion Language Models

Anonymous ACL submission

Abstract

Diffusion language models (DLMs) offer a parallel-decoding alternative to autoregressive generation, and sparse mixture-of-experts (MoE) variants such as the LLaDA series scale quality while keeping per-token compute low. On commodity GPUs, however, masked diffusion repeats a full-sequence forward pass across dozens of denoising steps and each step independently reloads expert weights from host memory, making generation prohibitively slow. We show that this cost is largely avoidable since adjacent denoising steps exhibit strong *temporal locality*–hidden-state cosine similarity above 0.97 and routed-expert Jaccard overlap above 0.78–which we exploit through three coordinated mechanisms: (i) per-expert *output caching* with cosine-similarity gating that skips expert work for stable tokens; (ii) *temporal weight prefetching* that uses previous-step routing to predict the next step's expert set; and (iii) a CPU SwiGLU *fallback* that absorbs misses by sending small activations to the host rather than pulling large weights back. Evaluated across two LLaDA-mini models, two GPU architectures (L40S and A6000), and seven benchmarks spanning code, mathematical, and general reasoning, our combined caching delivers a median **10× end-to-end speedup** over an HF Accelerate baseline at matched memory budgets, with accuracy preserved within sampling noise.

1 Introduction

DLMs generate by iterative denoising rather than left-to-right decoding, offering parallel token generation and bidirectional context; recent mixture-of-experts (MoE) variants (Bie et al., 2025, 2026) extend this paradigm to 16 B-parameter models while activating only ∼1.4 B parameters per token. Combining parallel decoding with sparse compute is well suited precisely to the resource-constrained settings where it is hardest to run. MoE routing sends each token to a small subset of experts (top-k, with k typically 1–8 of dozens to hundreds of experts per layer), so the experts any single step touches are few; but the full expert set is large, and on commodity GPUs it does not fit in VRAM. Profiling shows that streaming these experts from host memory, not compute, dominates inference time.

Masked diffusion compounds this cost. Unlike autoregressive (AR) decoding, which extends a sequence one token at a time, masked DLMs run 32–64 denoising steps over the *same* sequence, each performing a full forward pass (Bie et al., 2025, 2026). Inference systems built for AR generation–MoE-Infinity (Xue et al., 2024), FIDDLER (Kamahori et al., 2025), and HuggingFace Accelerate's `device_map="auto"`–treat each step independently; under a 12–24 GB consumer-GPU budget they synchronously demand-load expert weights on every step, paying the full PCIe cost dozens of times per generation.

However, the same iterative structure creates an opportunity that AR generation does not– consecutive denoising steps reprocess the same sequence with only small perturbations, both the experts that fire and the outputs they produce are highly redundant across steps, which we exploit in our runtime.

Contributions. We present an integrated runtime built on three temporal-locality mechanisms (i) per-expert output cache with cosine-similarity gating that classifies each token into reuse, partial-reuse, or recompute; avoids touching expert weights for the dominant reuse class (ii) temporal weight prefetching that uses the prior step's routing as the prediction target, staged across a pool of GPU buffers with independent transfer streams a configurable number of layers ahead of compute (iii) a CPU SwiGLU fallback that handles residual misses by sending ∼32 KB of activations to the host instead of pulling 6.3 MB of weights across PCIe

We evaluate across two task domains (code

generation, reasoning), two model versions (LLaDA2.0-mini, LLaDA2.1-mini) and two GPU architectures (L40S Ada Lovelace, A6000 Ampere) at a fixed 12 GB GPU memory. Combined caching delivers a median 10× speedup over HF Accelerate with accuracy preserved within sampling noise, demonstrating generalization across models, hardware generations, and tasks.

2 Related Work

MoE inference for autoregressive models. Production MoE stacks accelerate token-by-token decoding by managing where experts live and when they load: activation-trace–guided cache replacement and prefetching (MoE-Infinity (Xue et al., 2024)), CPU execution of mispredicted experts (FIDDLER (Kamahori et al., 2025)), multi-granularity mixed-precision caching (HOBBIT (Tang et al., 2024)), decoupled selection-execution to overlap loads with compute (Pre-gated MoE (Hwang et al., 2024)), and fine-grained or affinity-aware offloading (FineMoE (Yu et al., 2025), ExFlow (Yao et al., 2024)). None exploit the multi-step temporal locality of diffusion inference, which AR decoding does not produce.

Reducing expert activations. TEAM (Wei et al., 2026) observes the same temporal–spatial routing consistency in MoE DLMs but targets a different bottleneck: with the full model resident in VRAM (single A100 80 GB), it activates fewer experts per forward pass to cut *computation* — a lossy change to routing. We instead cut *I/O at a constrained memory budget* (12 GB), eliminating PCIe transfers without modifying routing. Foundational sparse-activation work (Shazeer et al., 2017; Fedus et al., 2022) likewise alters computation; we remove only the I/O cost of executing it.

Diffusion-specific inference. dLLM-Cache (Liu et al., 2025) caches intermediate computations across denoising steps but for *dense* (non-MoE) diffusion LMs. dInfer (Ma et al., 2025) targets multi-GPU decoding throughput, not single-GPU memory pressure. DiT-MoE (Fei et al., 2024) scales sparse diffusion transformers to vision with much shorter sequences. To our knowledge, ours is the first runtime to combine cross-step output reuse, cross-step weight prefetching, and host-resident compute fallback for MoE diffusion LMs.

3 Motivation

Where MoE-DLM inference time goes. MoE-DLMs combine top-k routing over 64–256 experts per layer (typically k=8) with 32–64 denoising passes over the same sequence: per-pass compute is sparse, yet repeated dozens of times per generation. Profiling a forward pass of LLaDA2.1-mini at MoE-layer granularity (CUDA-event timing on H100), the expert-FFN component dominates, accounting for **87–89%** of layer execution time–set by per-expert kernel launches when token counts are small, and by PCIe traffic when expert weights are streamed from host. Because masked diffusion repeats this forward pass at every denoising step, the bottleneck recurs dozens of times per generation. We next analyze the structural properties of iterative denoising to ask how much of that repeated cost is recoverable.

Profiling Observations. We instrument 20 MoE layers of LLaDA2.1-mini (256 experts, top-8, BF16) with forward pre-hooks recording the layer-input hidden states and the gate's top-k expert indices, over a 64-step generation (gen-length 64, 32 steps per 32-token block) at each denoising step. Averaging across all layers and consecutive step pairs (definitions in Appendix A):

1. **Routing stability.** Jaccard overlap of the active expert sets at each layer between adjacent steps, averages to **78–84%**, with low across-layer variance. Roughly four of five experts active at one step remain active at the next. *Previous-step routing is a high-accuracy, parameter-free predictor of next-step routing.*

2. **State similarity.** Per-token cosine similarity between input hidden states delivered to each layer at consecutive denoising steps, averages to ≥ 0.97 throughout generation, including tokens that are being actively denoised. *Per-expert outputs barely change between steps and can be reused rather than recomputed.*

3. **Expert sparsity.** The per-expert token-assignment distribution at every layer and denoising step is heavy-tailed: top 10 experts per layer absorb ∼**65%** of total token assignments, while **178 of 256** experts process zero tokens per step. *The active expert set per step is small, concentrated, and (by Obs. 1) temporally stable.*

4 Methodology

Based on the redundancies revealed in Section 3, we propose – **per-expert output caching** to exploit state similarity, and **temporal expert weight prefetching** with a CPU-resident SwiGLU fallback to exploit routing stability and sparsity – integrated into a **combined three-tier architecture** that runs both within a single forward pass.

4.1 Per-Expert Output Caching

A lightweight per-layer cache ($\sim$40MB) stores the outputs, routing IDs, and input hidden states of top-k experts. At step $t{+}1$, tokens are categorized using cosine-similarity threshold τ:

- **Cat A (Full Reuse)**: Same experts, similar input ($\cos > \tau$). Cached outputs are re-weighted by current gate; *no expert work.*
- **Cat B (Partial Reuse)**: Similar input, routing partially changed. Reuses shared experts and recomputes only newly-routed ones.
- **Cat C (Full Recompute)**: Significant input change. Standard MoE forward pass.

At $\tau{=}0.99$ with naive reuse, the cache eliminates expert work for 62–85% of tokens (Section 6), removing both PCIe traffic and compute.

4.2 Temporal Expert Weight Prefetching

Expert weights ($\sim$32 GB for LLaDA2.1-mini) exceed the 12–24 GB consumer GPU budget and reside in pinned host memory; individual experts are copied to device over PCIe on demand. Let $E_\ell^t \subseteq \{1, \ldots, 256\}$ denote the set of experts selected for at least one token at MoE layer ℓ and step t (typically $|E_\ell^t| \approx 60$–80). Every uncached expert in E_ℓ^{t+1} places its PCIe transfer on the compute critical path, stalling layer ℓ's matmul. We instead issue transfers on independent CUDA streams ahead of time, using E_ℓ^t to predict E_ℓ^{t+1}—accurate because $|E_\ell^t \cap E_\ell^{t+1}|/|E_\ell^{t+1}| \geq 0.78$ (Section 3). Figure 1 illustrates the resulting three-tier pipeline.

Temporal predictor. We take $\hat{E}_\ell^{t+1} \stackrel{\text{def}}{=} E_\ell^t$: experts routed at layer ℓ during step t become the prefetch target for the same layer at step $t{+}1$. The predictor has no learned parameters and reuses only the previous step's routing decisions. Experts in $E_\ell^{t+1} \setminus \hat{E}_\ell^{t+1}$ are residual misses handled by the fallback below.

Multi-stream buffer pool with layer-level lookahead. The prefetch path maintains N on-device buffers $\{\mathcal{B}_1, \ldots, \mathcal{B}_N\}$, each holding up to $C{=}256$ experts ($C \cdot W_e \approx 1.6$ GB; $W_e \approx 6.3$ MB per expert). N is set by GPU memory remaining after non-expert state: at a 12 GB budget, $N{=}5$, consuming $\approx$8 GB. A single *compute stream* σ_c runs attention, gating, the fused MoE matmuls, and unsort; N *transfer streams* $\sigma_t^1, \ldots, \sigma_t^N$ are bound one-to-one to buffers, so the DMA engine interleaves up to N layers' host-to-device copies rather than serializing them behind σ_c. A lookahead D specifies how many MoE layers are staged ahead of compute. At each denoising step, the scheduler assigns a free buffer to each of the next $\min(D, N)$ layers and enqueues, on that buffer's transfer stream, the copies for $\hat{E}_\ell^{t+1}$, recording a CUDA event when layer ℓ's copies have all been issued. When σ_c reaches layer ℓ, it issues $\sigma_c.\text{wait}(\text{evt}[\ell])$ before dispatching matmuls—a GPU-side dependency, not a host synchronization. After layer ℓ completes, its buffer is reassigned to layer $\ell{+}D$ on the same transfer stream. The matmul itself uses a two-launch fused SwiGLU kernel (gate, up, SiLU, and elementwise multiply in one Triton kernel; down-projection in a second), eliminating the per-expert launch overhead that dominates in the small-token regime. When $D > N$ no free buffer exists at assignment time and the scheduler preempts the buffer of the layer with the largest scheduling slack; the preempted transfer is reissued or, if compute begins first, served via the fallback. Section 7 characterizes this tradeoff. The per-step PCIe volume is $\sum_\ell |\hat{E}_\ell^{t+1}| \cdot W_e$, determined entirely by the temporal predictor.

CPU SwiGLU fallback. A residual miss at layer ℓ is an expert $e \in E_\ell^{t+1} \setminus \hat{E}_\ell^{t+1}$; synchronously fetching its 6.3 MB weight would add $\sim$0.35 ms to the critical path. The fallback inverts the transfer direction: the activation tile for e ($|\text{tokens}_e| \cdot H \cdot 2$ B, $\approx$32 KB at typical token counts) is copied to host, SwiGLU runs on the pinned host weights via a CPU thread pool, and outputs are copied back. Because per-miss volume is $\sim$200$\times$ smaller than the corresponding weight transfer, the round-trip is issued on a transfer stream and overlaps with the remainder of σ_c.

The per-step prefetch–compute pipeline pseudocode is given in Algorithm 1 (Appendix B).

4.3 Combined Three-Tier System

The output cache (Section 4.1) and temporal weight prefetching (Section 4.2) are complementary: the

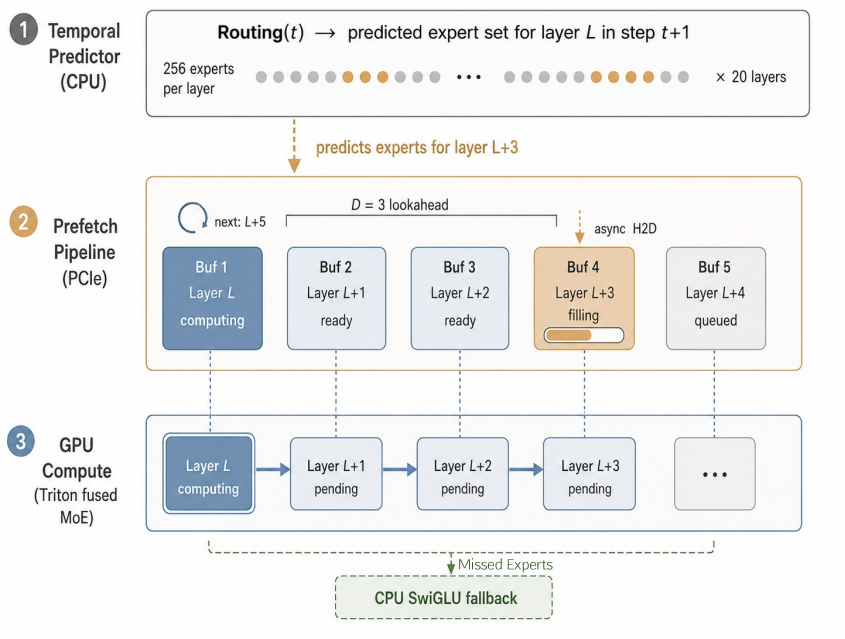

Figure 1: Temporal expert weight prefetching. The CPU-side predictor uses the previous step's per-layer routing E_ℓ^t to forecast $\hat{E}_\ell^{t+1}$ and issues asynchronous host-to-device copies on independent transfer streams into a pool of on-device buffers, each staged D layers ahead of the GPU compute stream. A buffer cycles back into the pool immediately after its layer completes, allowing PCIe transfers to overlap with compute in steady state.

cache eliminates both computation and weight access for Cat A tokens, while prefetching reduces expert-weight latency for Cat B and Cat C tokens.

We integrate both mechanisms into a unified forward pass, summarized in Figure 2. For each MoE layer at denoising step $t+1$:

- **Cat A (Full Reuse):** Tokens are served entirely from cache; expert weights are never touched.
- **Cat B (Partial Reuse):** Shared experts reuse cached outputs, while newly-routed experts are served from the prefetched GPU buffer, falling back to CPU SwiGLU on miss.
- **Cat C (Full Recompute):** All experts are executed through the prefetched GPU path or CPU fallback.

Crucially, the cache also reshapes the prefetching problem itself: Category A experts become invisible to the temporal predictor because they no longer require expert execution. As a result, the prediction set shrinks substantially, allowing the prefetcher to concentrate its limited on-device buffer capacity on the small subset of experts that genuinely require transfers. This tighter prediction window both reduces aggregate PCIe traffic and improves effective prediction accuracy.

5 Experimental Setup

5.1 Models

We evaluate two MoE DLMs from the LLaDA family — LLaDA 2.0-mini (Nie et al., 2025) and LLaDA 2.1-mini (Bie et al., 2026) — to validate that our system is model-version-agnostic. Both share the same sparse-MoE architecture (256 experts per MoE layer, top-8 routing, 20 layers, hidden 2048, intermediate 512, BF16), totaling ∼16 B parameters with ∼1.4 B active per token.

5.2 Hardware

We evaluate on two distinct GPU architectures to test hardware agnosticism: **L40S** (Ada Lovelace, 48 GB GDDR6, 864 GB/s bandwidth, 91 BF16 TFLOPS) and **A6000** (Ampere, 48 GB GDDR6, 768 GB/s bandwidth, 38 BF16 TFLOPS). To simulate a low-resource consumer-GPU environment, we cap usable VRAM at 12 GB (enforced via Accelerate's per-device memory cap), forcing a 63% offload ratio. Host RAM (∼120 GB) is more than enough for pinned expert weights.

5.3 Benchmarks

We span four task domains, chosen to stress both the headline speedup claim and its generalization across very different generation regimes (long-form

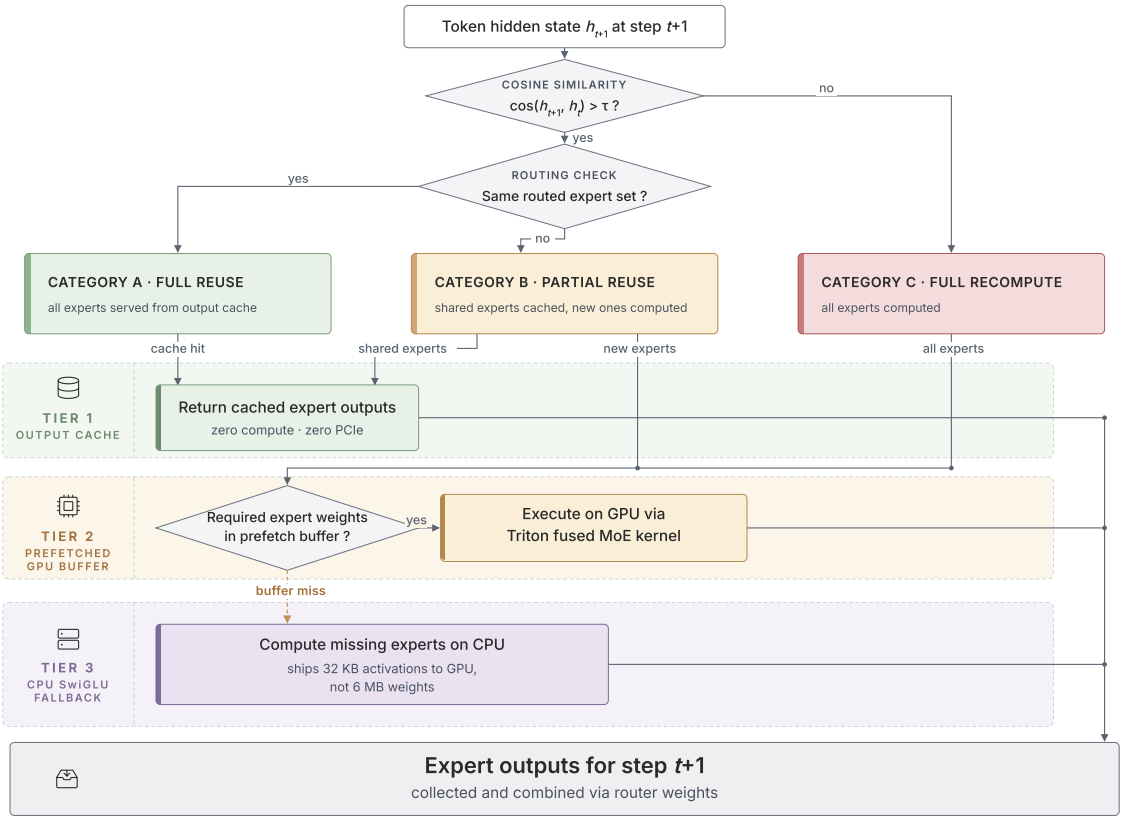

Figure 2: Combined three-tier dispatch at step $t+1$. Each token is classified using hidden-state cosine similarity and routing-set overlap, producing Cat A (full cache reuse), Cat B (partial reuse), or Cat C (full recompute). Cat B/C computation first attempts execution from the prefetched GPU expert buffer via the fused Triton MoE kernel, while residual misses are handled through the CPU SwiGLU fallback.

code, short MCQ answers, math chain-of-thought, and multi-step compositional reasoning).

Code generation (long-context). *HumanEval+* (Liu et al., 2023) (164 Python function-completion problems with augmented unit tests; pass@1) and *LiveCodeBench (easy+medium)* (Jain et al., 2024) (310 contamination-resistant competitive-programming problems; pass@1).

General knowledge. *ARC-C* (Clark et al., 2018) (1,172 grade-school science MCQ, 4 options; accuracy/letter match); *MMLU* (Hendrycks et al., 2021) (MCQ across 57 academic and professional subjects; accuracy/letter match); *TriviaQA* (Joshi et al., 2017) (1,000 open-domain factoid questions; free-form answer scored by exact-match and token-F1).

Mathematical reasoning. *GSM8K* (Cobbe et al., 2021) (1,319 grade-school math problems with multi-step CoT; exact-match on numerical answer).

Long compositional reasoning. *BBH* (Suzgun et al., 2023) (6,511 reasoning examples across 23 compositional subtasks; per-subtask exact-match / accuracy).

Per-benchmark prompting protocols, generation lengths, and scoring details are given in Appendix D.

5.4 Methods Compared

On every benchmark, we compare two end-to-end serving strategies at fixed 12 GB GPU memory:

HF Accelerate baseline. HF Accelerate with automatic device placement, demand-loading non-resident expert weights synchronously over PCIe at every denoising step — the practitioner's default for memory-constrained MoE serving. Standard MoE inference frameworks (e.g. MoE-Infinity, FIDDLER, HOBBIT) are tuned for autoregressive serving and do not translate to the multi-step, single-sequence structure of masked-diffusion in-

ference; HF Accelerate is the only drop-in baseline for this regime.

Combined three-tier system. Our full system (Section 4.3): per-expert output caching, temporal weight prefetching with the CPU SwiGLU fallback, and the fused Triton MoE kernel, integrated into a single forward pass. End-to-end speedup is the ratio of accelerate to combined per-problem wall time.

Intermediate prefetch mode (HumanEval+, GSM8K). On these two benchmarks we additionally report the prefetch-only configuration (no output cache) to verify the additive speedup progression: accel $\ll$ prefetch $<$ combined. The same ordering is implicit on the other benchmarks since the cache is purely additive over prefetch.

5.5 Preliminary Ablations

Three hyperparameters — cosine-similarity threshold τ, prefetch lookahead D, and routing predictor — were fixed by ablations prior to the main evaluation (Appendix C): (i) naive reuse at τ=0.99 gives the largest speedup at no accuracy cost (tighter thresholds and the Jacobian variant lose net throughput); (ii) D=3 minimises p50 ms/step at every (model, GPU-budget) cell, with deeper lookahead adding 9–37% latency without raising hit rate; (iii) the pure-temporal predictor outperforms spatial, deep, and EMA-workload augmentations, which raise hit rate by 1–4 pp but slow ms/step by 3–64%.

5.6 Configuration

Generation runs at temperature 0 with top-k=8 routing, 32 diffusion steps, and block length 32. Following the ablations of Section 5.5, the output cache uses naive reuse at τ=0.99, the prefetcher uses lookahead D=3 over a 5-buffer pool, and routing predictions come from previous-step routing. Code execution for coding tasks runs in a sandboxed subprocess with a 60s timeout per problem.

6 Results

We report end-to-end speedup of the combined system over the accelerate baseline (Section 5.4) across all benchmarks, models and GPUs, along with the per-benchmark quality metric (Section 5.3). On HumanEval+ and GSM8K we also compare intermediate prefetch mode to verify the additive speedup progression: prefetching is faster than the accelerate baseline, and combined caching is faster than prefetching.

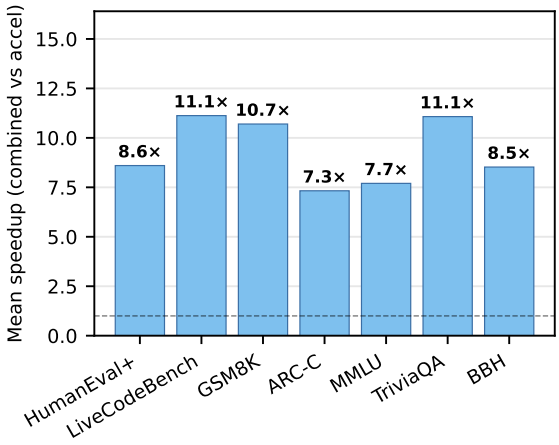

Figure 3: Mean speedup with combined caching (combined vs accel) per benchmark at the 12 GB budget, averaged over 2 models $\times$ 2 GPUs. *All seven benchmarks land in a 7–11$\times$ mean-speedup band with combined caching.*

6.1 End-to-End Speedup

Figure 3 summarises the combined/accelerate speedup per benchmark, averaged over the four (model, GPU) cells. Every benchmark sits between 7$\times$ and 11$\times$ on average, indicating that the system delivers a large and consistent end-to-end win across very different generation regimes.

Table 1 reports the accel/prefetch/combined progression on L40S for HumanEval+ and GSM8K: prefetching alone delivers a 4–7$\times$ improvement over accelerate, and the output cache layered on top extends this to 8–11$\times$, confirming the additive speedup predicted by the design. Table 2 then consolidates the combined/accelerate ratio across the full benchmark suite, (model, GPU) cells; the median across cells is approximately **10**$\times$, with the largest gains on the longer-context LCB and TriviaQA benchmarks, consistent with the higher cache-hit fraction at longer generations (Section 6.2).

Quality preservation. Across every (model, GPU) cell, output quality with combined caching tracks the accelerate baseline within $\leq$4 pp on the coding benchmarks (HumanEval+ pass@1, LCB pass@1) and within $\leq$1.5 pp on GSM8K exact-match (90.0% vs 90.1% on LLaDA 2.0-mini; 85.6% vs 84.2% on LLaDA 2.1-mini); the full per-cell table is reported in Appendix F. The same no-degradation conclusion is reported by prior work (Liu et al., 2025, Table 1).

Table 1: Per-step latency (ms) and speedup over accelerate on L40S at 12 GB. For each model: top row ms/step, bottom row speedup over accel. *Each layer is additive: prefetching delivers 4–7× over accel, and the output cache layered on top extends this to 8–11× on both models and both benchmarks.*

Benchmark	Model	accel	prefetch	combined
HumanEval+	2.0-mini	5086	758	612
		1.0×	6.7×	8.3×
	2.1-mini	4764	751	585
		1.0×	6.3×	8.1×
GSM8K	2.0-mini	3931	790	362
		1.0×	5.0×	10.9×
	2.1-mini	2959	735	363
		1.0×	4.0×	8.2×

Table 2: Combined / accelerate end-to-end speedup across benchmarks and (model, GPU) cells at the 12 GB memory budget. *Median speedup over HF Accelerate across cells: ∼10×.*

	LLaDA 2.0-mini		LLaDA 2.1-mini	
Benchmark	**L40S**	**A6000**	**L40S**	**A6000**
HumanEval+	8.3×	12.4×	8.1×	5.6×
LiveCodeBench	11.5×	11.3×	9.2×	12.5×
GSM8K	10.9×	12.4×	8.2×	11.3×
ARC-C	5.5×	9.8×	5.6×	8.4×
MMLU	9.7×	10.3×	5.6×	5.2×
TriviaQA	7.0×	14.4×	14.3×	8.6×
BBH	14.2×	5.8×	5.0×	9.1×

6.2 Cache Mechanism Across Benchmarks

Figure 4 reveals the source of the speedup variation across domains: the cache-hit (Cat A) fraction varies from 64% on HumanEval+ up to 88% on MMLU/GSM8K. These high Cat A fractions reveal that across every benchmark a substantial fraction of expert recomputation can be avoided whenever consecutive denoising steps see nearly-identical inputs. Cat B is consistently <2% across all benchmarks: at τ=0.99, tokens that change routing have almost always also drifted enough in input space to fail the cosine threshold, falling through to Cat C, which absorbs the remainder. Exact per-benchmark percentages are tabulated in Appendix E.

6.3 GPU Agnosticism

Different (GPU, benchmark) cells obtain varying degrees of speedup, reflecting interactions between GPU compute throughput, PCIe bandwidth, benchmark-specific generation length, and the resulting per-cell cache-hit rate. Across all cells, however, both GPUs consistently deliver substantial speedups with no exceptions. The Cat A fraction at fixed model and benchmark also matches within 0.5 pp between L40S and A6000, confirming that the cache mechanism is a property of the diffusion process rather than of the underlying hardware.

7 Analysis

The system is PCIe-bandwidth-bound, not latency-bound. The lookahead sweep (Appendix C.2, Table 5) shows that increasing D from 3 to 12 adds 9–37% wall time; D=3 is optimal across all configurations. Two measurements pin down the saturation point. (i) At D=3 the prefetcher already transfers ≈18.7 GB per denoising step on the 12 GB-budget runs (≈24 GB/s sustained), near the PCIe Gen4 ceiling on L40S/A6000; deeper D does not retire transfers any faster, it only queues more pending DMAs and (when $D>N=5$) triggers buffer stealing. (ii) At the 24 GB budget the hit rate is exactly flat across $D \in \{3, 6, 12\}$ for llada21-mini: the larger buffer pool has nothing additional to retire, independently confirming that D=3 already covers the in-flight PCIe budget. The speedup comes from cutting per-step PCIe volume by ∼75%, not from hiding latency. As a sanity check, $18.7\ \text{GB}/24\ \text{GB/s} \approx 780$ ms matches the 735–790 ms/step *prefetch* column of Table 1 within ∼6% across all four (benchmark, model) cells. The *combined* column (362–612 ms) is *not* on this curve: output-cache reuse strips ∼88% of per-step transfers (Cat A bypasses PCIe entirely), dropping per-step volume to ≈2.3 GB and shifting the bottleneck to compute and non-MoE layers.

Why pure temporal prediction is the default. We evaluated three augmented predictors against pure temporal (Appendix C.3, Table 6). *Temporal-spatial* adds the previous-step routing of adjacent layers ($\ell\pm1$) and halves the block-boundary miss spike (8.4% vs 16.2% miss rate at step 0). *EMA-workload-expanded* adds the top-K experts ranked by an exponentially-weighted moving average of per-step token counts and cuts the steady-state miss rate by 30% (6.9% vs 10.9% median). *Temporal-deep* adds the same layer's routing from two steps back and helps neither cold start nor steady state appreciably. None of the three eliminates the boundary spike, which occurs when 32 fresh tokens enter the active set with no routing history, and all three add 3–64% ms/step. Pure temporal is the default because reducing miss rate alone does not move end-

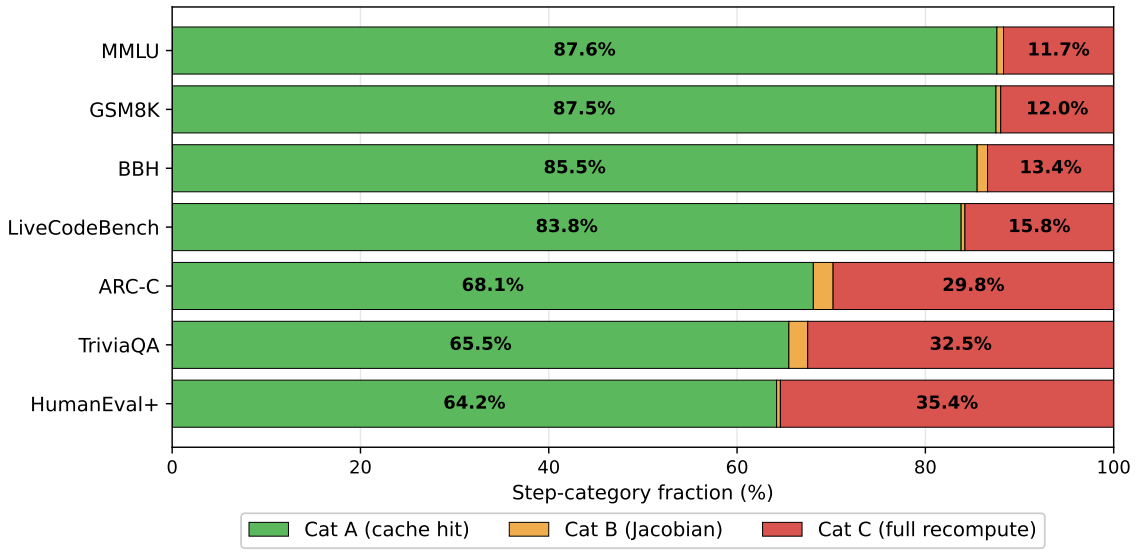

Figure 4: Cache-category breakdown under combined caching, averaged across (model, GPU) cells for each benchmark. Cat A: full output-cache reuse; Cat B: partial reuse; Cat C: full recompute. *Longer-generation and chain-of-thought benchmarks (LCB, BBH, GSM8K, MMLU) reach 84–88% cache hits per step; benchmarks with shorter or more discrete outputs (HumanEval+, TriviaQA, ARC-C) reach 64–68%.*

to-end latency in the PCIe-bound regime above; composing temporal-spatial with EMA-expanded is a natural extension we leave to future work.

Cat A reuse scales with generation length. Across our generation-length sweep (Appendix E, Table 7) Cat A grows with generation length: ∼67% at gen 128 (ARC-C, TriviaQA), 87% at 384 (GSM8K, MMLU), and 84–86% at 512–1024 (BBH, LCB). Longer generations contain more diffusion iterations whose consecutive steps are highly correlated, exposing more cacheable work, with the output cache paying off most on long-form natural-language generation (math, MCQ, factual reasoning).

Cat B (Jacobian correction) is unnecessary at τ=0.99. Cat B (input similar but routing changed) accounts for ≤2.2% of steps across all 7 cells, since tokens that change routing have nearly always also drifted in input space and fall into Cat C. We therefore disable the Jacobian correction path by default. The τ sweep (Appendix C.1, Table 3) confirms the operating point: tightening to τ=0.999 shrinks Cat A by ∼10 pp at no accuracy benefit, and enabling the Jacobian variant runs 2× slower than the naive path.

8 Conclusion

We presented a system for accelerating MoE diffusion LM inference by exploiting two properties unique to iterative denoising: cross-step routing stability and high cosine similarity of intermediate states. Combined caching delivers a median **10**× end-to-end speedup over HF Accelerate at 12 GB budgets with no quality degradation. Three findings generalize: (i) in this PCIe-bound regime the dominant lever is bytes transferred per step, not latency; (ii) residual misses are structural, concentrating at block boundaries where routing shifts wholesale, so no history-based augmentation anticipates them; (iii) caching benefit scales with generation length (20 pp higher Cat A at gen=1024 vs gen=512), making the approach increasingly valuable for longer-context tasks.

Limitations

Our evaluation is single-GPU and single-sequence; scaling temporal prefetching to batched or distributed serving (e.g., dInfer (Ma et al., 2025) with tensor parallelism) requires managing concurrent routing predictions and per-sequence cache pools, which we leave to future work. The techniques also target memory-constrained settings: on GPUs that hold the full MoE in VRAM, the prefetch and fallback paths are inert and only the output cache yields residual savings. Finally, we evaluate two domains (coding, reasoning) across three benchmarks. Extending to retrieval-style tasks, conversational generation, and tool-use settings is the next step.

References

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, Lun Du, Daozhuo Feng, Haibo Feng, Mingliang Gong, Zhuocheng Gong, Yanmei Gu, Jian Guan, Kaiyuan Guan, Hongliang He, Zenan Huang, Juyong Jiang, Zhonghui Jiang, Zhenzhong Lan, and 31 others. 2026. Llada2.1: Speeding up text diffusion via token editing. *Preprint*, arXiv:2602.08676.

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, and 12 others. 2025. Llada2.0: Scaling up diffusion language models to 100b. *Preprint*, arXiv:2512.15745.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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

Zhengcong Fei, Mingyuan Fan, Changqian Yu, and Junshi Huang. 2024. Scaling diffusion transformers to 16 billion parameters. *arXiv preprint arXiv:2407.11633*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations (ICLR)*.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. 2024. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *International Symposium on Computer Architecture (ISCA)*.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*.

Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (ACL)*.

Keisuke Kamahori and 1 others. 2025. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations (ICLR)*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Zhiyuan Liu, Yicun Yang, Yaojie Zhang, Junjie Chen, Chang Zou, Qingyuan Wei, Shaobo Wang, and Linfeng Zhang. 2025. dllm-cache: Accelerating diffusion large language models with adaptive caching. *Preprint*, arXiv:2506.06295.

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, and 4 others. 2025. dinfer: An efficient inference framework for diffusion language models. *Preprint*, arXiv:2510.08666.

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

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

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics (ACL)*.

Peng Tang and 1 others. 2024. HOBBIT: A mixed precision expert offloading system for fast MoE inference. *arXiv preprint arXiv:2411.01433*.

Linye Wei, Zixiang Luo, Pingzhi Tang, and Meng Li. 2026. TEAM: Temporal-spatial consistency guided expert activation for MoE diffusion language model acceleration. *arXiv preprint arXiv:2602.08404*.

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

Jinghan Yao, Quentin Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. 2024. Exploiting inter-layer expert affinity for accelerating mixture-of-experts model inference. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*.

Hanfei Yu and 1 others. 2025. FineMoE: Taming latency-memory trade-off in MoE-based LLM serving via fine-grained expert offloading. *arXiv preprint arXiv:2502.05370*.

A Profiling Definitions

This appendix gives precise definitions for the three quantities measured in Section 3.

A.1 Notation

We adopt the following notation throughout, instantiated for LLaDA2.1-mini in the profiling setup of Section 3:

- L — number of MoE layers (L=20).
- E — number of experts per MoE layer (E=256).
- k — routing fan-out (top-k experts per token, k=8).
- T — number of denoising steps per generation (T=64 in the profiling setup; the main experiments of Section 5 use T=32, the model's recommended configuration).
- S — sequence length (prompt plus generation tokens).
- d — model hidden dimension (d=2048).
- $\ell \in \{1, \ldots, L\}$ — MoE layer index.
- $t \in \{0, \ldots, T-1\}$ — denoising step index.
- $H_\ell^t \in \mathbb{R}^{S \times d}$ — input hidden states to MoE layer ℓ at step t, recorded by a forward pre-hook on the MoE block.
- $R_\ell^t \in \{1, \ldots, E\}^{S \times k}$ — top-k expert indices selected by the gate of layer ℓ at step t for each token.
- $E_\ell^t = \{R_\ell^t[s, j] : s \in [S], j \in [k]\} \subseteq \{1, \ldots, E\}$ — the set of *active* experts at layer ℓ, step t (experts selected for at least one token).

A.2 Routing Stability (Observation 1)

The Jaccard overlap between active expert sets at consecutive steps is

$$J_\ell^t = \frac{\left|E_\ell^t \cap E_\ell^{t+1}\right|}{\left|E_\ell^t \cup E_\ell^{t+1}\right|} \in [0, 1].$$

The reported 78–84% range is the per-layer mean $\bar{J}_\ell = \frac{1}{T-1} \sum_{t=0}^{T-2} J_\ell^t$, taken across all L MoE layers, with the spread reporting min and max over ℓ.

A.3 State Similarity (Observation 2)

For each token position s, the per-token cosine similarity at layer ℓ between consecutive steps is

$$c_\ell^t[s] = \frac{\langle H_\ell^t[s,:],\, H_\ell^{t+1}[s,:] \rangle}{\left\|H_\ell^t[s,:]\right\|_2 \cdot \left\|H_\ell^{t+1}[s,:]\right\|_2} \in [-1, 1].$$

The reported ≥ 0.97 value is the grand mean

$$\bar{c} = \frac{1}{L\,S\,(T-1)} \sum_{\ell=1}^{L} \sum_{s=1}^{S} \sum_{t=0}^{T-2} c_\ell^t[s],$$

computed in float32 to avoid BF16 dot-product rounding error.

A.4 Expert Sparsity (Observation 3)

For each expert $e \in \{1, \ldots, E\}$, the per-expert token-assignment count at layer ℓ and step t is

$$n_\ell^t[e] = \left|\{(s, j) : R_\ell^t[s, j] = e\}\right|.$$

The *top-10 token-mass share* ρ_ℓ^t is the sum of the ten largest entries of $n_\ell^t[\,\cdot\,]$ divided by the total mass $\sum_{e=1}^{E} n_\ell^t[e]$; the *idle expert count* is $m_\ell^t = \left|\{e : n_\ell^t[e] = 0\}\right|$. The reported ∼65% and 178/256 values are averages of ρ_ℓ^t and m_ℓ^t over all layers and denoising steps.

B Prefetch Algorithm

Algorithm 1 gives the full pseudocode for the per-step prefetch–compute pipeline described in prose in Section 4.2, covering the initial lookahead staging, the per-layer compute sweep with stream synchronization, and buffer recycling for layers $\ell+D$ onward.

C Preliminary Ablations

This appendix reports the ablation data summarised in Section 5.5. Each subsection establishes the operating point for one hyperparameter; the corresponding interpretation appears in Section 7.

C.1 Output Cache: τ Sweep

We compared the output cache at three cosine-similarity thresholds, with and without the Jacobian Taylor correction, on a tuning subset of MMLU and GSM8K. We tune on this subset rather than the coding benchmarks to fix τ on data disjoint from the headline coding/reasoning evaluation; the resulting operating point (τ=0.99) transfers without modification (Section 6). Table 3 reports speedup and

Algorithm 1 Per-step prefetch–compute pipeline (denoising step $t{+}1$).

Require: hidden states H_{t+1}; MoE layers $\ell_1 < \cdots < \ell_L$; buffers $\mathcal{B}_1, \ldots, \mathcal{B}_N$; streams $\sigma_c, \sigma_t^1, \ldots, \sigma_t^N$; previous routing E_ℓ^t; lookahead D.

for $i = 1$ to $\min(D, N)$ **do** ▷ stage initial lookahead
  $\ell \leftarrow \ell_i$; assign $\mathcal{B}_i$ to ℓ; $P_\ell \leftarrow E_\ell^t$
  for $e \in P_\ell$ **do**
    `cudaMemcpyAsync`(cpu$[\ell, e] \rightarrow$ slot in $\mathcal{B}_i$) on σ_t^i
  end for
  record event `evt`$[\ell]$ on σ_t^i
end for
for $j = 1$ to L **do** ▷ compute sweep
  $\ell \leftarrow \ell_j$; $\mathcal{B} \leftarrow$ buffer assigned to ℓ
  attention, gating on $\sigma_c \Rightarrow$ `topk_idx`; $E_\ell^{t+1} \leftarrow$ unique(`topk_idx`)
  σ_c.`wait`(`evt`$[\ell]$) ▷ GPU-side wait
  for $e \in E_\ell^{t+1} \setminus$ loaded$(\mathcal{B})$ **do** ▷ residual miss
    sync-load e into $\mathcal{B}$, **or** CPU SwiGLU(e)
  end for
  sort tokens by expert; build offsets; dispatch fused MoE on σ_c
  release $\mathcal{B}$; assign to ℓ_{j+D}; start prefetch on $\sigma_t^{\mathrm{idx}(\mathcal{B})}$
  $E_\ell^t \leftarrow E_\ell^{t+1}$ ▷ for next step's prediction
end for

Table 3: Output cache: speedup and accuracy at varying cosine-similarity thresholds, on the MMLU/GSM8K tuning subset. Naive reuse at $\tau{=}0.99$ delivers the largest speedup at no accuracy cost. Accuracies are subset values and are not directly comparable to the full-set results of Section 6.

Config	Speedup	MMLU	GSM8K
Baseline	1.00×	100.0%	96.7%
naive $\tau{=}0.99$	**2.3–2.8**×	100.0%	96.7%
naive $\tau{=}0.995$	1.62×	100.0%	96.7%
naive $\tau{=}0.999$	1.34×	100.0%	86.7%
Jacobian $\tau{=}0.99$	0.49×	100.0%	86.7%

Table 4: Token-category distribution by τ (naive reuse, tuning subset). Tightening τ from 0.99 to 0.999 shifts ∼6 pp from Cat A reuse to Cat C full recomputation without measurable accuracy benefit. *Hit*: fraction of expert evaluations served from cache; *Skip*: fraction of per-expert weight loads avoided.

τ	A	B	C	Cold	Hit	Skip
0.99	72.4%	2.4%	16.4%	8.7%	73.7%	74.4%
0.995	70.5%	1.3%	19.5%	8.7%	69.8%	71.6%
0.999	66.8%	0.2%	24.4%	8.6%	64.5%	67.0%

downstream accuracy; Table 4 reports the resulting token-category distribution.

Operating point. We adopt naive reuse at $\tau{=}0.99$. Tighter thresholds force tokens into full recomputation without quality benefit; the Jacobian correction (three additional matmuls per Cat B token) costs more than it saves.

C.2 Prefetch Lookahead Depth

We swept the layer-level lookahead $D \in \{3, 6, 12\}$ across both models and three GPU memory budgets on the full GSM8K test set (1,319 problems, L40S). The 16 and 24 GB budgets are included to confirm that $D{=}3$ is optimal not only under the 12 GB target but as memory is relaxed—i.e., the operating point is not a 12 GB artifact. Results in Table 5.

Table 5: Lookahead depth ablation on GSM8K (p50 ms/step). $D{=}3$ is optimal at every (model, GPU-budget) configuration, including as the budget is relaxed beyond the 12 GB target.

Model	GPU	$D{=}3$	$D{=}6$	$D{=}12$
LLaDA2.0-mini	12 GB	**790**	918	1074
	16 GB	**783**	835	1034
	24 GB	**791**	820	955
LLaDA2.1-mini	12 GB	**735**	865	992
	16 GB	**744**	770	932
	24 GB	**735**	776	875

Operating point. We adopt $D{=}3$ with a 5-buffer pool. Increasing D from 3 to 12 adds 9–37% latency while changing hit rate by at most 1–2 pp; the reason is analysed in Section 7.

C.3 Prediction Strategy

We compared the pure-temporal predictor (previous-step routing) against three augmented variants (LLaDA2.1-mini, L40S, 12 GB). This setup isolates prediction quality from prefetch-pool sizing, which the lookahead sweep (Appendix C.2) isolates separately. Results in Table 6.

Operating point. We adopt the pure-temporal

Table 6: Prediction strategy ablation. Pure temporal sits on the Pareto frontier: every augmented predictor raises hit rate by 1–4 pp but slows ms/step. *Cap*: maximum number of experts prefetched per layer per step.

Policy	Cap	ms/step	Δ ms	Hit %
Temporal (ours)	160	**672–693**	ref	89–91%
Temporal-deep (D=2)	180	690–752	+1–8%	90–94%
EMA workload (K=8)	180	693–738	+3–6%	93–94%
Temporal-spatial	200	1100–1155	+64%	91–95%

predictor. The augmented predictors enlarge the prefetched set and therefore the PCIe volume per step, which dominates the small hit-rate gains.

D Benchmark Details

This appendix expands on the benchmark summaries in Section 5.3, giving the full per-benchmark prompting protocol, generation length where applicable, and scoring rule.

HumanEval+ (Liu et al., 2023). 164 Python function-completion problems with ∼80× more unit tests per problem than the original HumanEval, making the benchmark substantially more robust to surface-level pattern matching. Generation length 512 tokens, average ∼215 tokens emitted per problem. Zero-shot instruction-tuned chat prompting; scored by unit-test pass rate (pass@1).

LiveCodeBench (easy+medium) (Jain et al., 2024). 310 contamination-resistant competitive-programming problems drawn from LeetCode, AtCoder, and Codeforces. Generation length 1024 tokens, average ∼310 tokens emitted. Zero-shot instruction-tuned chat prompting; scored by unit-test pass rate (pass@1).

ARC-C (Clark et al., 2018). 1,172 grade-school science multiple-choice questions with four options each, selected to resist surface-pattern shortcuts and to require commonsense plus scientific knowledge. Zero-shot instruction prompting → letter answer.

MMLU (Hendrycks et al., 2021). Multiple-choice questions across 57 subjects spanning STEM, humanities, social sciences, and applied/professional domains, designed to assess breadth of academic and professional knowledge. 4-shot chain-of-thought prompting → letter A/B/C/D.

TriviaQA (Joshi et al., 2017). 1,000 open-domain factoid questions drawn from trivia leagues and quiz bowls. 4-shot string prompting → free-form short answer; scored by exact-match and token-F1 against an alias set per question.

GSM8K (Cobbe et al., 2021). 1,319 grade-school math word problems requiring multi-step arithmetic and chain-of-thought reasoning. Generation length 384 tokens, 4-shot CoT prompting; scored by exact-match on the final numerical answer.

BBH (Suzgun et al., 2023). 6,511 reasoning examples across 23 subtasks (yes/no, multiple-choice letter, free-form text, integers, lists) drawn from BIG-Bench and selected to target compositional and multi-step reasoning. Zero-shot chain-of-thought prompting; per-subtask scoring rules follow the original release.

E Cache-Category Breakdown

Table 7 reports the exact per-benchmark Cat A/B/C fractions averaged over GPU cells; the same numbers are visualised in Figure 4. The category fractions are essentially GPU-invariant (matches within 0.5 pp between L40S and A6000).

Table 7: Cache-category breakdown under combined caching, averaged across (model, GPU) cells. Cat A: full output-cache reuse; Cat B: partial reuse; Cat C: full recompute.

Benchmark	Cat A	Cat B	Cat C
HumanEval+	64.2%	0.4%	35.4%
LiveCodeBench	83.8%	0.4%	15.8%
GSM8K	87.5%	0.5%	12.0%
ARC-C	68.1%	2.1%	29.8%
MMLU	87.6%	0.7%	11.7%
TriviaQA	65.5%	2.0%	32.5%
BBH	85.5%	1.1%	13.4%

F Quality Preservation

Table 8 reports the per-cell quality metric for each measured (benchmark, model, GPU) cell, comparing the accelerate baseline against combined caching. The metric is pass@1 for the code benchmarks (HumanEval+, LCB), letter-match accuracy for the MCQ benchmarks (ARC-C, MMLU), and exact-match accuracy on the parsed answer for the remaining free-form benchmarks (TriviaQA, GSM8K, BBH); the exact prompting protocols and scoring rules are in Section 5.3 and Appendix D. Grouping by benchmark family, the largest combined–accel gap is ≤3.5 pp on code,

$\leq$2.8 pp on free-form recall (TriviaQA), $\leq$2.2 pp on MCQ and compositional reasoning (ARC-C, MMLU, BBH), and $\leq$1.5 pp on math (GSM8K) for full-set cells; the A6000/2.0-mini GSM8K cell uses an n=30 accel sub-sample (caption), so its 6.3 pp gap is a noisy point estimate rather than a measured degradation. No cell shows a sizeable quality loss.

Table 8: Per-cell output quality at the 12 GB budget. pass@1 for HumanEval+ (n=164) and LCB (n=310); letter-match accuracy for ARC-C (n=1,172) and MMLU (n=14,042); exact-match accuracy for TriviaQA (n=1,000), GSM8K (n=1,319), and BBH (n=6,511). GSM8K accel values are sub-sampled (n=300 on L40S; n=30 on A6000) because the accel baseline runs prohibitively slowly at this memory budget; the matched-subset point estimate is reported.

		LLaDA 2.0-mini		**LLaDA 2.1-mini**	
Benchmark	**GPU**	accel	combined	accel	combined
HumanEval+	L40S	82.9	80.5	81.7	82.9
	A6000	79.9	80.5	82.9	79.9
LCB	L40S	49.4	49.7	39.4	42.9
	A6000	47.1	48.1	41.3	41.9
GSM8K	L40S	90.1	90.0	84.2	85.6
	A6000	96.7	90.4	83.3	84.5
ARC-C	L40S	89.0	88.5	87.6	87.8
	A6000	88.9	88.9	87.7	87.2
TriviaQA	L40S	44.3	44.4	43.2	44.6
	A6000	46.2	43.4	44.2	44.8
MMLU	L40S	74.4	76.6	58.0	58.2
	A6000	70.3	68.9	54.2	52.7
BBH	L40S	50.7	49.2	48.8	48.6
	A6000	49.5	50.3	49.1	49.2

G Potential Risks

Our system accelerates inference of existing publicly released MoE-DLMs without altering their parameters or routing; combined caching preserves output quality within sampling noise (Appendix F), so we introduce no new failure modes or biases beyond those already present in the underlying models. The principal consideration is dual-use: lowering the memory floor for capable MoE-DLMs widens access for research and education but also marginally lowers the barrier to misuse. The increment is small relative to existing access via larger GPUs or hosted APIs.

H Ethics Statement

We do not train or fine-tune any model, collect or annotate any data, or involve human subjects. Experiments use publicly released benchmarks and pre-trained LLaDA weights via HuggingFace. AI assistants were used for writing and code review; all technical content was authored and verified by the listed contributors.

I Artifact Licenses and Intended Use

HumanEval+, GSM8K, MMLU, and BBH are released under MIT; LiveCodeBench under CC BY 4.0; ARC-C under CC BY-SA 4.0; TriviaQA under Apache 2.0. LLaDA 2.0-mini and 2.1-mini are distributed through HuggingFace under their model-card licenses. Our implementation builds on HuggingFace Transformers, Accelerate, PyTorch, and Triton, with the HumanEval+ harness derived from `evalplus` (Apache 2.0). All artifacts are used for inference benchmarking only, consistent with each artifact's intended use.