BEYOND FORWARD COUNTS: CACHING IN MOE DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Caching reduces repeated token computation in diffusion language models, but its serving benefit in offloaded mixture-of-experts (MoE) models depends on how reuse propagates through expert execution and weight movement. These operations occur at different granularities: routed query positions are grouped into expert calls, while expert residency determines which weights must move. Consequently, a large reduction in processed query positions need not produce a comparable reduction in expert calls or transferred bytes, and forward count alone cannot characterize serving efficiency. We introduce TRACE-MoE, an execution-aware evaluation framework that jointly measures query visits, forward evaluations, expert calls, transferred bytes, and request latency to connect token reuse with its downstream execution costs. Family-matched controls and paired timing compare cache policies under identical requests and runtime configurations, while independent diagnostic runs reconcile the measured work counters. Workload-preserving controls provide a timing reference, and fixed-forward-count comparisons reveal changes in work within denoising steps. Controlled executions of Fast-dLLM and dKV-Cache on LLaDA2 mini checkpoints compare cache policies with their corresponding family controls at a common 64-token generation budget. These measured configurations include lower latency with more forward evaluations, with paired savings of up to 59.0% when forward count increases. TRACE-MoE makes the relationships among token reuse, expert demand, weight movement, and latency explicit, providing a practical basis for evaluating cache policies under matched execution conditions.

1 INTRODUCTION

Diffusion language models (DLMs) generate multiple token positions in parallel and iteratively refine a partially masked sequence through denoising (Sahoo et al., 2024; Nie et al., 2025; Ye et al., 2025). Unlike autoregressive decoding with key/value (KV) caching, diffusion inference repeatedly revisits intermediate representations, creating opportunities for computation reuse. Training-free acceleration techniques, such as Fast-dLLM and dKV-Cache, exploit this property by reusing KV representations across denoising steps, reducing redundant computation without additional model training (Wu et al., 2025b; Ma et al., 2025). Other approaches extend reuse to adaptive token selection and hierarchical block caches (Jiang et al., 2025; Lee et al., 2026; Wu et al., 2025a).

MoE diffusion models such as LLaDA2.0 and LLaDA2.1 bring sparse routing into this iterative generation path (Bie et al., 2025; 2026). Expert placement and execution scheduling then become central to serving efficiency (Rajbhandari et al., 2022; Kamahori et al., 2024; Xue et al., 2024). In expert-offloaded execution, query positions are routed and grouped by expert, and each demanded expert is served from device memory or loaded from host memory. Figure 1 traces this execution path from cache-dependent query processing to expert calls, weight transfers, and request latency. The key distinction is that **query processing, expert execution, and weight movement occur at different granularities**. Reusing some query positions can leave the same experts demanded by the remaining positions, while the weights that must move depend on expert residency. Token reuse therefore changes several execution costs whose reductions need not be proportional.

Prior diffusion-cache studies establish the value of reuse through generation quality and measured inference speed (Wu et al., 2025b; Ma et al., 2025; Jiang et al., 2025). For expert-offloaded MoE diffusion models, understanding that benefit requires following the avoided token computation

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

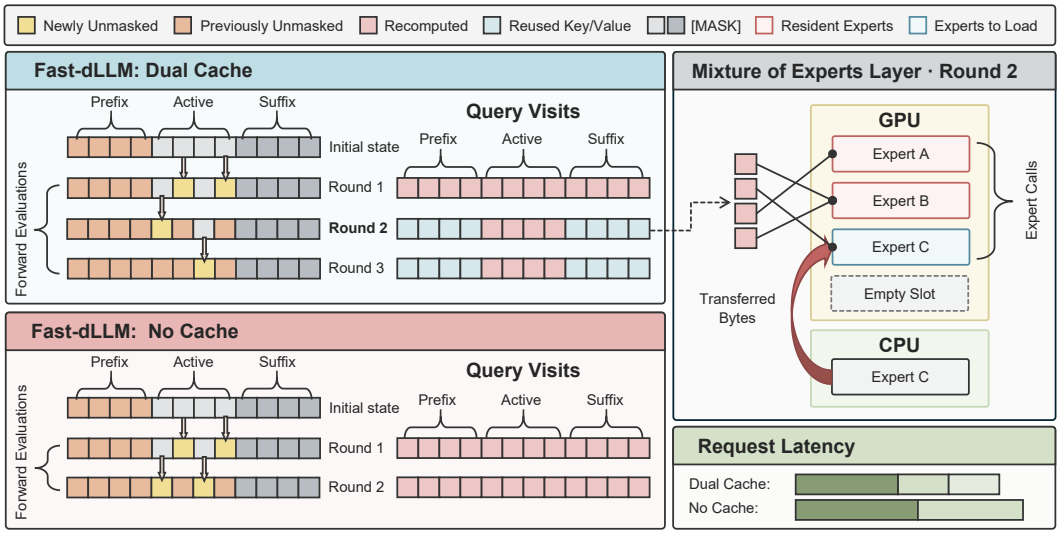

Figure 1: From token reuse to serving cost in expert-offloaded diffusion inference. The schematic contrasts Fast-dLLM dual caching with no cache and connects denoising rounds and query visits to expert calls, CPU-to-GPU weight transfers, and request latency.

through routing and expert residency. Studies of memory-aware kernels and offloaded serving likewise show why arithmetic work and data movement must be considered together (Dao et al., 2022; Sheng et al., 2023). The number of forward evaluations (NFE) records how often the model runs, but the work performed within each forward depends on the active query positions, demanded experts, and resident weights. Consequently, we ask: **how does token reuse translate into serving cost across these execution granularities?**

To answer this question, we introduce TRACE-MoE (Token Reuse And Cost Evaluation for MoE), an execution-aware evaluation framework for expert-offloaded diffusion inference. TRACE-MoE characterizes each request using a unified cost vector (Q, N, C, B, T): query visits, forward evaluations, expert calls, transferred bytes, and end-to-end latency. This joint accounting connects cache-induced changes in query processing with the expert calls and weight transfers that remain during execution, and relates those changes to observed request latency.

Three design choices make these measurements interpretable. **Family-matched controls** compare each cache policy with its corresponding control under identical requests and runtime configurations. **Paired timing and independent diagnostics** address measurement order and counter consistency: timing pairs reverse the mode order, and separate diagnostic runs reconcile query assignments, expert calls, and weight transfers. **Workload-preserving and fixed-NFE comparisons** provide complementary references. The former establish a timing reference when measured work is unchanged; the latter reveal reductions in the work performed within denoising steps. These choices connect policy-induced work changes to latency while keeping the measurement conditions explicit.

We instantiate TRACE-MoE on LLaDA2 mini checkpoints with Fast-dLLM and dKV-Cache. Across the measured configurations, query visits, expert calls, and transferred bytes exhibit distinct reductions under caching. The controlled executions also include lower latency with increased NFE, with savings of up to 59.0% relative to the matching family control, alongside savings at fixed NFE. These measurements relate differences in execution granularity to the observed serving behavior and motivate considering both forward count and the work performed within each forward. We report latency at a common 64-token generation budget and record task correctness at both 64 and 512 tokens for the inputs in Table 2, with each result tied to its request, cache mode, and generation budget.

Our contributions are as follows:

- **An execution-level view of cache efficiency.** We connect query processing, expert demand, and expert residency to explain why token reuse and forward count capture different aspects of serving cost.
- **TRACE-MoE: an execution-aware evaluation framework.** We jointly measure query visits, forward evaluations, expert calls, transferred bytes, and latency using family-matched controls, paired timing, and independent diagnostic checks.
- **Controlled evidence across execution regimes.** We use workload-preserving controls and comparisons with fixed or varying forward counts to characterize how cache-induced work reductions translate into request latency.

2 Related Work

Diffusion generation and caching. Discrete and masked diffusion establish iterative text generation (Austin et al., 2021; Lou et al., 2024; Sahoo et al., 2024), while block-based models support KV reuse across generated blocks (Arriola et al., 2025; Bie et al., 2025; 2026). Fast-dLLM combines approximate KV caching with confidence-aware decoding (Wu et al., 2025b); dKV-Cache delays and conditions key/value reuse (Ma et al., 2025). Adaptive token selection (Jiang et al., 2025; Lee et al., 2026), hierarchical caches (Wu et al., 2025a), and wavefront decoding (Tian et al., 2026) further reduce recomputation or increase parallelism. We study how cache-induced work changes propagate to expert execution and weight movement.

Sparse expert execution and offloading. Sparse routing expands model capacity through conditional computation (Shazeer et al., 2017; Lepikhin et al., 2020; Fedus et al., 2022; Jiang et al., 2024). DeepSpeed-MoE, Tutel, and MegaBlocks optimize expert execution, dispatch, and sparse kernels (Rajbhandari et al., 2022; Hwang et al., 2023a; Gale et al., 2023). Offloaded inference adds residency and transfer decisions, addressed through predictive gating, CPU–GPU coordination, and expert caching (Hwang et al., 2023b; Kamahori et al., 2024; Xue et al., 2024). In diffusion serving, query reuse changes the expert-demand stream, making processed positions, expert calls, and weight loads distinct accounting units.

Execution-aware evaluation. The Roofline model and IO-aware attention distinguish arithmetic work from data movement (Williams et al., 2009; Dao et al., 2022); PagedAttention and FlexGen expose the role of memory management in generation (Kwon et al., 2023; Sheng et al., 2023). Benchmarking also requires specified workloads, quality targets, and accounting for measurement variability (Reddi et al., 2020; Kalibera & Jones, 2013). We apply these principles through joint (Q, N, C, B, T) accounting, family-matched controls, and paired timing, connecting token reuse to observed request cost.

3 Motivation

Observation 1: token reuse and serving cost have different granularities. Query visits count the positions processed across denoising steps, whereas NFE counts how often the model runs. Neither quantity specifies the expert work performed within those forwards. An MoE dispatcher groups routed positions into expert calls (Fedus et al., 2022; Rajbhandari et al., 2022): removing some positions can shrink a group without eliminating its expert call. The weights that must move additionally depend on whether the demanded experts are resident on the GPU. Thus, query visits, expert calls, and weight transfers can respond differently to the same reuse policy. The measurement problem is to connect these execution events to the latency of a complete request.

Observation 2: more forwards can coexist with less work and lower latency. Figure 2 makes this distinction concrete under family-matched controls at a 64-token generation budget. On M20/GSM8K, Fast dual increases NFE from 29 to 36, yet reduces query visits by 73.6%, expert calls by 48.5%, transferred bytes by 68.3%, and request latency by 46.7%. Forward count alone would miss the saving because it gives each forward equal weight despite changes in the work inside it. Query-visit reduction alone would also overstate the reduction in expert calls. This example motivates accounting for both the number of forwards and their execution work, rather than treating either as a sufficient proxy for request cost.

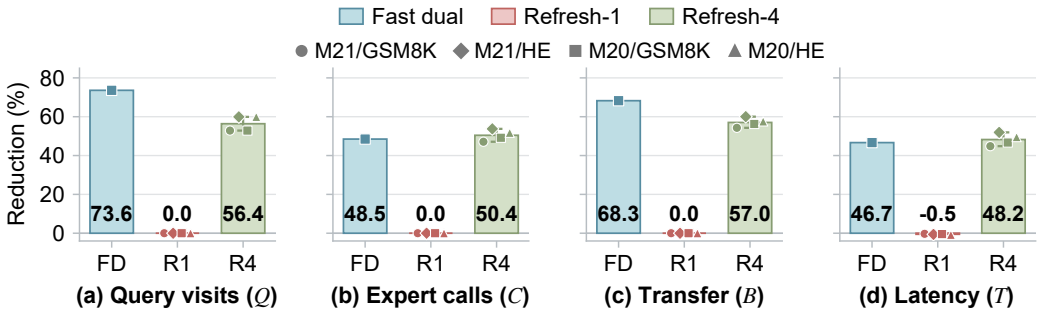

Figure 2: Execution-cost reductions relative to family-matched controls. FD: Fast dual on M20/GSM8K (NFE $29 \rightarrow 36$); R1/R4: dKV Refresh-1/4 (NFE $64 \rightarrow 64$). Bars and base labels show the FD value or dKV mean across evaluated settings; point shapes identify settings. Capped lines show the Refresh-4 min–max range.

Observation 3: equal forward counts can conceal different execution costs. The dKV comparisons in Figure 2 hold NFE at 64 across checkpoint–request settings. Refresh-1 matches disabled dKV in query visits, expert calls, and transferred bytes, with a mean paired latency saving of -0.5%. Refresh-4 preserves the same forward count but reduces those work terms by 56.4%, 50.4%, and 57.0% on average, alongside a mean paired latency saving of 48.2%. These controls answer complementary questions: Refresh-1 supplies a timing reference when measured work is unchanged, while Refresh-4 reveals savings within a fixed forward schedule. Their comparison shows why enabling a cache and counting forwards are insufficient to establish what execution work was avoided.

From these comparisons to an evaluation contract. These comparisons motivate connecting the positions reused to the expert calls and weight loads that remain, together with latency measured under controlled executions. Joint (Q, N, C, B, T) accounting makes these changes explicit. Family-specific controls preserve the decoding rule used for each comparison; workload-preserving and fixed-NFE controls distinguish timing variation at unchanged work from changes within forwards. Section 4 implements these requirements in TRACE-MoE through a shared execution contract, paired timing, and independent diagnostic checks.

4 TRACE-MoE: Execution-Aware Evaluation Framework

4.1 System architecture and execution contract

TRACE-MoE follows each request–mode pair through a complete execution pipeline: *request adapter* $\rightarrow$ *cache controller* $\rightarrow$ *denoising executor* $\rightarrow$ *router/dispatcher* $\rightarrow$ *residency manager and offloader* $\rightarrow$ *GPU expert execution* $\rightarrow$ *cost ledger*. The request adapter fixes the prompt, model checkpoint, generation budget, and runtime configuration. The cache controller applies the family-specific policy, while the executor advances the denoising state. Routing and residency are recorded at the point where they affect expert calls and host-to-device movement; the ledger then joins these counters with the end-to-end timer.

Formally, let $r = (p, b, \theta)$ denote a request with prompt p, generation budget b, and checkpoint θ. At forward t, the execution state is

$$s_t = (X_t, \mathcal{K}_t, \mathcal{H}_t), \tag{1}$$

where X_t is the denoising token state, $\mathcal{K}_t$ is the logical KV state, and $\mathcal{H}_t = \{\mathcal{H}_{\ell,t}\}_\ell$ is the set of GPU-resident experts at each MoE layer. A cache policy π_m maps the state to active positions, reusable positions, and a refresh decision:

$$(A_t, U_t, a_t) = \pi_m(s_t, t), \qquad q_t = |A_t|. \tag{2}$$

The denoising executor then applies the model transition

$$(X_{t+1}, \mathcal{K}_{t+1}) = \Phi_\theta(X_t, \mathcal{K}_t; A_t, U_t, a_t, \mathcal{H}_t). \tag{3}$$

This contract keeps policy decisions separate from their execution consequences: π_m selects reuse, while Φ_θ determines which routed experts are actually invoked and which weights must move.

For layer ℓ, let $R_{\ell,t}(u) \subseteq \mathcal{E}_\ell$ be the experts selected by the router for active position u. Their union forms the demanded expert set $\mathcal{E}_{\ell,t}$. The residency manager derives the miss set and updates the resident set as

$$\begin{aligned} \mathcal{E}_{\ell,t} &= \bigcup_{u \in A_t} R_{\ell,t}(u), \qquad \mathcal{M}_{\ell,t} = \mathcal{E}_{\ell,t} \setminus \mathcal{H}_{\ell,t}, \\ \mathcal{H}_{\ell,t+1} &= \mathrm{Keep}_K(\mathcal{H}_{\ell,t} \cup \mathcal{M}_{\ell,t}), \end{aligned} \tag{4}$$

where K is the per-layer resident-expert capacity and Keep_K denotes the runtime's eviction and retention rule, including access order. The main runtime serves resident experts first, so initially resident demanded experts are used before any eviction; under this ordering, the miss set is exactly the set difference above. Appendix D formalizes the resulting coverage and residency bounds.

4.2 Joint accounting of work and latency

For each request–mode execution, TRACE-MoE records the cost vector $\mathbf{x}_{r,m} = (Q, N, C, B, T)$. If forward t processes q_t active query positions and the run executes N forwards, query visits are

$$Q = \sum_{t=1}^{N} q_t. \tag{5}$$

This quantity counts repeated processing of positions across denoising steps. The generation budget separately specifies the number of output positions.

Let $\mathcal{E}_{\ell,t}$ be the set of experts demanded at MoE layer ℓ in forward t. In the evaluated runtime, each demanded expert is invoked once on its group of assigned positions. Thus, expert calls satisfy

$$C = \sum_{t=1}^{N} \sum_{\ell} |\mathcal{E}_{\ell,t}|. \tag{6}$$

For a weight payload of size $w_{\ell,e}$ bytes, the host-to-device transfer volume is

$$B = \sum_{t=1}^{N} \sum_{\ell} \sum_{e \in \mathcal{M}_{\ell,t}} w_{\ell,e}, \tag{7}$$

where $\mathcal{M}_{\ell,t} = \mathcal{E}_{\ell,t} \setminus \mathcal{H}_{\ell,t}$ denotes the experts required at layer ℓ during forward t but absent from the GPU-resident set before execution. Latency is measured by a request-level wall-clock timer:

$$T = t_{\mathrm{complete}} - t_{\mathrm{start}}. \tag{8}$$

In the main evaluation, the timer starts before logical cache reset and ends after generation, decoding, and CUDA completion, including prompt transfer. Model loading, weight preparation, warm-up, diagnostic materialization, and scoring occur outside this boundary. The cost vector thus links four execution counters to observed request latency: Q tracks positions, N counts forward transitions, C tracks demanded experts, and B tracks loads determined by residency.

4.3 Family-specific controls and paired timing

Fast-dLLM compares no cache, prefix cache, and dual cache (Wu et al., 2025b). dKV-Cache compares disabled caching, refresh at every step (Refresh-1), and refresh every four steps (Refresh-4) (Ma et al., 2025). Main timing trials use BF16 weights, eager execution, a 64-token generation budget, and the same resource configuration. The logical expert cache starts cold for each trial.

For each mode m, TRACE-MoE pairs R primary repeats with the corresponding family control 0 on identical request tokens and runtime settings. Let $x_{m,i,k}$ be metric k for repeat i and define the reduction metrics $\mathcal{K}_\downarrow = \{Q, C, B, T\}$. The paired estimator is

$$\Delta_{m,k} = \frac{1}{R} \sum_{i=1}^{R} \left(1 - \frac{x_{m,i,k}}{x_{0,i,k}}\right), \quad k \in \mathcal{K}_\downarrow, \qquad \Gamma_{m,N} = \frac{1}{R} \sum_{i=1}^{R} \left(\frac{N_{m,i}}{N_{0,i}} - 1\right). \tag{9}$$

For the main evaluation, $R = 2$ and $S_m = \Delta_{m,T}$ is the paired latency saving. It averages within-pair ratios rather than taking the ratio of mean latencies. Mode order is reversed between the two repeats. These paired savings use the control from the same cache family, so a Fast mode is normalized to Fast uncached and a dKV mode to dKV disabled. This normalization measures each policy's effect relative to its own decoding rule. Positive $\Delta_{m,k}$ denotes a reduction, while positive $\Gamma_{m,N}$ denotes an increase in NFE.

The ledger associates each measurement with a record identity

$$e = (r, m, \rho, \chi, \lambda), \tag{10}$$

where ρ identifies the source report, χ identifies the configuration and source hash, and λ records the run status. Request/control matching, counter reconciliation, repeat metadata, and completed-run status connect each figure and table value to its source execution record.

4.4 Diagnostic checks and budget-specific quality

Primary trials in the main evaluation collect shape and offloader counters. Separate diagnostic runs capture forward and route information, checking repeatability and agreement between token assignments, expert calls, cache misses, and transfer bytes. This separation keeps diagnostic tensor materialization outside the primary timing path. The result ledger also retains source hashes and the identity of each input and configuration.

Task correctness is recorded at 64 and 512 tokens for the inputs in Table 2. Each run retains all generated positions and scores the prefix through the first end-of-sequence token. These request-level scores are indexed by generation budget in Appendix B. We also use offline trajectory screens to examine how specific refresh and residency rules affect query visits and cache misses on archived traces.

5 Evaluation

5.1 Experimental Setup

Models and requests. The controlled executions use LLaDA2.1-mini (M21) and LLaDA2.0-mini (M20) (Bie et al., 2026; 2025). Tables 1–2 use GSM8K and HumanEval inputs (Cobbe et al., 2021; Chen et al., 2021), with identical tokenized inputs across checkpoints and cache modes; an M20 execution extension uses GSM8K, HumanEval, and MATH-500 inputs listed in Appendix B. Scoring uses numeric exact match for GSM8K and functional execution for HumanEval.

Cache families and controls. Fast-dLLM is evaluated with its no-cache, prefix-cache, and dual-cache modes. dKV-Cache is evaluated with disabled caching, Refresh-1, and delayed Refresh-4. Each cached mode is paired with the control from its own family, so the comparison keeps the decoding rule and request tokens fixed while changing the reuse policy.

Environment and measurement protocol. The main evaluation uses one NVIDIA A800-SXM4-80GB with a 22 GiB device-allocation budget, BF16 weights, eager execution, and 48 expert slots per MoE layer. The offloader prioritizes resident experts and loads missing weights from pinned host memory. Generation uses block length 16 and temperature zero; Fast uses a confidence threshold of 0.95, while dKV uses the fixed refresh schedules defined in Section 4. The timing protocol pairs cache modes with their family controls, reverses mode order across repeats, and separates warm-up and diagnostic records from timed executions.

Cost and quality operating points. The 64-token trials record (Q, N, C, B, T) under a common generation horizon, pairing each cache mode with its family control. For the inputs in Table 2, task correctness is recorded at 64 and 512 tokens using the scoring rules above. Appendices B and F report budget-specific outcomes and additional execution regimes, respectively.

Table 1: Execution-work ledger for the M21 and M20 settings, relative to family-matched controls. Each setting reports reductions in query visits Q (%), expert calls C (%), and transferred bytes B (%), followed by the control→cache NFE transition. Values are rounded to one decimal except NFE. Bold marks the largest family-normalized reduction in each metric and setting.

	M21/GSM8K				M21/HE				M20/GSM8K				M20/HE			
Mode	$Q\downarrow$	$C\downarrow$	$B\downarrow$	NFE	$Q\downarrow$	$C\downarrow$	$B\downarrow$	NFE	$Q\downarrow$	$C\downarrow$	$B\downarrow$	NFE	$Q\downarrow$	$C\downarrow$	$B\downarrow$	NFE
dKV R4	52.9	47.1	54.3	64→64	59.9	53.7	60.0	64→64	52.9	**49.2**	56.3	64→64	**59.9**	**51.7**	**57.6**	64→64
Fast prefix	67.9	53.6	65.7	26→25	67.6	56.8	66.3	20→21	52.3	45.8	62.2	29→33	38.1	31.0	33.0	8→8
Fast dual	**76.7**	**61.1**	**71.4**	26→22	**74.6**	**60.7**	**70.0**	20→21	**73.6**	48.5	**68.3**	29→36	46.8	35.8	37.7	8→8

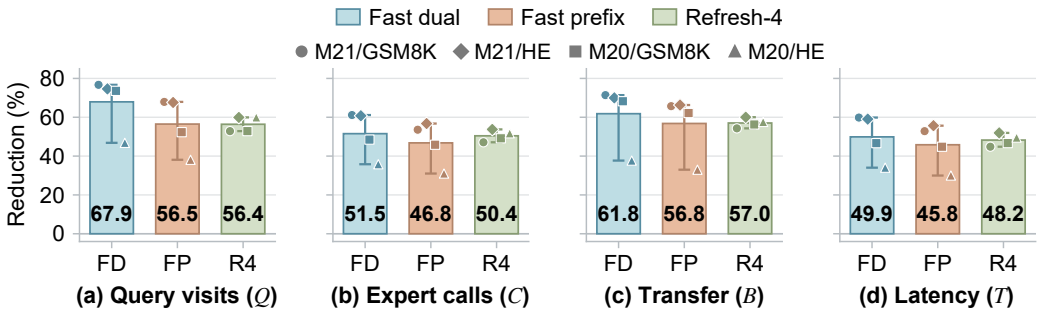

Figure 3: Execution-cost reductions for Fast dual (FD), Fast prefix (FP), and dKV Refresh-4 (R4), relative to family-matched controls. Bars and base labels report means; point shapes identify the evaluated settings. Capped lines show min–max ranges across settings.

5.2 Efficiency Evaluation

Query, expert, and transfer work. Table 1 reports work counters for the M21 and M20 settings; Figure 3 summarizes their means and ranges. **Query-visit reduction exceeds expert-call reduction in every row of Table 1.** Among these modes, delayed dKV Refresh-4 is the only one with a fixed $64 \rightarrow 64$ NFE transition in every setting; its query, expert, and transfer reductions remain in the 52.9–59.9%, 47.1–53.7%, and 54.3–60.0% ranges, respectively. Fast prefix and Fast dual reach as high as 67.9% and 76.7% query reduction, but their NFE transitions vary from a decrease to an increase (for example, $29 \rightarrow 36$ for M20/GSM8K under Fast dual). The ledger quantifies how large reductions in processed query positions translate into smaller, configuration-dependent reductions in demanded experts.

Matched query savings, different expert demand. Refresh-4 produces identical query-visit reductions across checkpoints for matched inputs: 52.9% on GSM8K and 59.9% on HumanEval. The corresponding expert-call reductions differ: 47.1% versus 49.2% on GSM8K, and 53.7% versus 51.7% on HumanEval, for M21 and M20 respectively. Thus, fixing both NFE and the fraction of query work removed does not fix the reduction in expert demand. The remaining query positions can cover different expert sets across checkpoints. The reversal of the checkpoint ordering across evaluated settings also indicates that a single conversion factor from query savings to expert-call savings would miss the structure of these measurements.

Transfer volume per expert call. The transfer ledger distinguishes fewer expert invocations from fewer weight loads per invocation. For M20/GSM8K, Fast dual retains 51.5% of the control's expert calls but 31.7% of its transferred bytes. The ratio B/C therefore falls by 38.4%, computed from the unrounded counters. With a common expert payload within this checkpoint, this ratio is proportional to the fraction of calls that require loading weights. Both fewer calls and a lower load fraction contribute to the observed transfer reduction. The resident capacity remains 48 experts per layer in both modes; the difference occurs along their respective execution traces and does not require allocating additional expert slots.

Table 2: Mean request latency (s) at the 64-token budget, with paired latency savings in parentheses. All values are rounded to one decimal; savings are computed before rounding. Bold entries mark the lowest latency within each cache family. HE denotes HumanEval.

	Fast-dLLM			dKV-Cache		
Setting	No cache	Prefix	Dual	Disabled	Refresh-1	Refresh-4
M21/GSM8K	25.2	11.9 (52.8%)	**10.1** (59.8%)	61.4	61.6 (−0.3%)	**33.9** (44.8%)
M21/HE	23.3	10.3 (55.7%)	**9.5** (59.0%)	75.0	75.4 (−0.6%)	**36.0** (51.9%)
M20/GSM8K	28.1	15.5 (44.8%)	**15.0** (46.7%)	63.4	63.7 (−0.5%)	**33.8** (46.7%)
M20/HE	9.6	6.7 (30.0%)	**6.3** (34.0%)	76.5	77.1 (−0.8%)	**38.6** (49.5%)

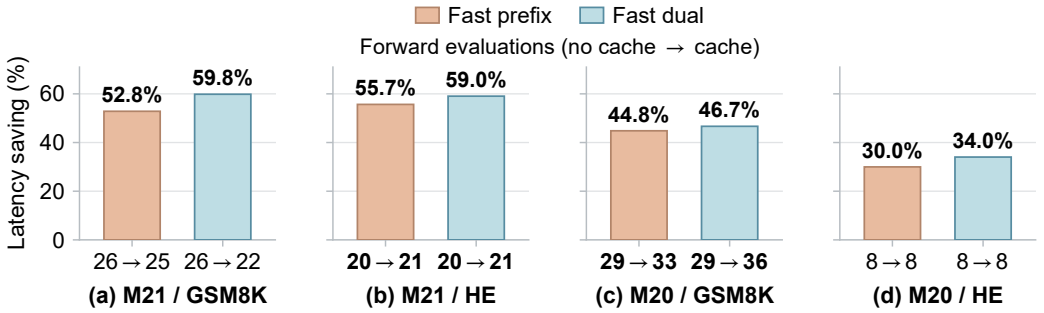

Figure 4: Fast-dLLM paired latency savings relative to no cache. Bar-top labels show savings; labels below the bars give forward evaluations (no cache→cache). Bold transitions indicate latency savings despite more forwards.

Latency under varying forward counts. Table 2 reports end-to-end request latency in seconds and the paired saving relative to each family-matched control. Fast prefix saves 30.0–55.7% latency and Fast dual saves 34.0–59.8% across the evaluated settings, with dual improving on prefix in every row. **dKV Refresh-4 saves** 44.8**–**51.9% **latency at fixed NFE**, preserving the 64-forward schedule of its control. Refresh-1 preserves the measured workload and yields paired savings of −0.8% to −0.3%, providing a near-zero timing reference. Each latency result is thus paired with its measured workload and family control.

Latency savings across NFE regimes. Figure 4 details Fast-dLLM's varying-forward behavior. The transitions beneath the bars show **positive latency savings with decreasing, increasing, and unchanged NFE**: Fast dual lowers NFE for M21/GSM8K, raises NFE for M21/HE and M20/GSM8K, and leaves NFE fixed for M20/HE. M20/GSM8K dual saves 46.7% latency despite the $29 \rightarrow 36$ NFE increase. Dividing the measured counters by NFE shows that its query visits, expert calls, and transferred bytes per forward fall by 78.7%, 58.5%, and 74.4%, respectively. In this measured configuration, lower work per forward more than offsets the increase in forward count. M20/HE also retains a 34.0% latency saving at unchanged NFE. These accounting ratios connect per-forward work to measured totals, while the request timer records end-to-end latency across compute, transfers, and runtime operations.

Additional reuse at matched NFE. The prefix–dual comparison separates additional reuse from changes in forward count on HumanEval: both modes execute 21 forwards on M21 and eight on M20. Relative to prefix, dual further reduces query visits by 21.6% and 14.1%, expert calls by 9.2% and 7.0%, and transferred bytes by 10.9% and 7.0%, respectively. Their mean request latencies fall from 10.3 to 9.5 seconds and from 6.7 to 6.3 seconds. In these controlled executions, dual further reduces work and latency at the same NFE as prefix. The smaller reduction in expert calls again distinguishes reuse of query positions from elimination of entire expert invocations.

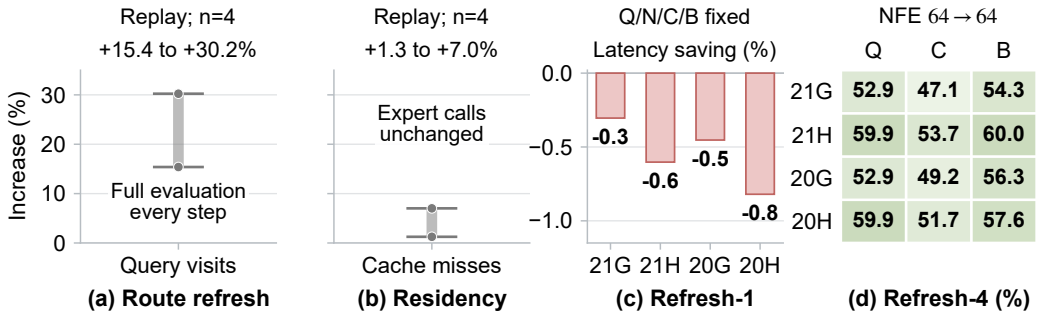

Figure 5: Replay diagnostics and live dKV controls. (a)–(b) Min–max increases in query visits and cache misses across evaluated replay settings; annotations describe the corresponding execution behavior. The replay panels report event counts under the specified controls. (c) Refresh-1 paired latency savings with unchanged $Q/N/C/B$. (d) Refresh-4 reductions (%) in $Q/C/B$ at NFE $64 \rightarrow 64$. 21/20 denote M21/M20; G/H denote GSM8K/HumanEval.

Additional M20 inputs. The extension in Table 3 yields paired latency savings of 20.4–56.3% for Fast prefix and 23.0–63.4% for Fast dual. On GSM8K/138, prefix increases NFE from 44 to 45 while reducing latency by 56.3%. Refresh-4 saves 46.3–48.6% at fixed $64 \rightarrow 64$ NFE across the extension inputs. These executions extend the increased- and fixed-NFE observations to additional inputs, with the corresponding query, expert-call, and transfer reductions reported alongside latency.

5.3 Control and Diagnostic Analysis

Preserved-work and fixed-NFE controls. Figure 5 contrasts two replay diagnostics (a)–(b) with live dKV controls (c)–(d). Refresh-1 provides a near-zero timing reference with unchanged $Q/N/C/B$, whereas Refresh-4 reduces query, expert, and transfer work at the same $64 \rightarrow 64$ NFE. The replay panels identify which work events change under the tested refresh and residency rules.

Replay diagnostics as mechanism tests. The replay panels compare query visits and cache misses across the evaluated replay settings. Relative to fixed-period refresh, the tested route-triggered rule performs full evaluation at every step and adds 15.4–30.2% query visits. Relative to the better of the tested hit-first and hot-set controls, refresh-aware residency retention adds 1.3–7.0% cache misses while leaving expert calls unchanged. These fixed-trace comparisons identify the execution events affected by each tested rule, with the schedules and residency controls specified in Appendix C.

The replay outcomes affect different entries in the ledger. Route-triggered refresh removes reuse steps, increasing query work; residency retention preserves expert calls but increases weight-loading events. By comparison, live Refresh-1 preserves both quantities, with latency changes below one percent. These observations locate each rule's effect in the ledger while keeping replay counts separate from live latency. Appendix E develops the resulting design implications.

6 Conclusion

We introduced TRACE-MoE to connect token reuse with serving cost in expert-offloaded diffusion language models. Its central principle is to account jointly for query visits, forward evaluations, expert calls, transferred bytes, and request latency, reflecting the different granularities of query processing, expert execution, and weight movement. Family-matched controls, paired timing, and independent diagnostics make these measurements comparable and interpretable. Workload-preserving controls establish a timing reference, while fixed-NFE comparisons reveal savings within denoising steps. In the measured configurations, query visits, expert calls, and transferred bytes exhibit distinct reductions, and lower latency coexists with more forward evaluations. By making these relationships measurable under matched requests, TRACE-MoE provides a practical basis for evaluating cache policies and locating opportunities to reduce expert execution and weight movement.

AI USE STATEMENT

During manuscript preparation, the authors used generative artificial intelligence (AI) tools for language editing, manuscript organization, figure and table refinement, interpretation of experimental results, reference checking, and LaTeX debugging. AI-assisted suggestions for presentation and analysis were checked against the source data and implementation by the authors. The authors take responsibility for the final text, figures, claims, and experimental results.

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. *arXiv preprint arXiv:2503.09573*, 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, et al. LLaDA2.0: Scaling Up Diffusion Language Models to 100B. *arXiv preprint arXiv:2512.15745*, 2025.

Tiwei Bie, Maosong Cao, Xiang Cao, Bingsen Chen, Fuyuan Chen, Kun Chen, et al. LLaDA2.1: Speeding Up Text Diffusion via Token Editing. *arXiv preprint arXiv:2602.08676*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359, 2022.

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.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. MegaBlocks: Efficient Sparse Training with Mixture-of-Experts. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 288–304, 2023.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, et al. Tutel: Adaptive Mixture-of-Experts at Scale. In *Proceedings of Machine Learning and Systems*, volume 5, pp. 269–287, 2023a.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pregated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference. *arXiv preprint arXiv:2308.12066*, 2023b.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, et al. Mixtral of Experts. *arXiv preprint arXiv:2401.04088*, 2024.

Yuchu Jiang, Yue Cai, Xiangzhong Luo, Jiale Fu, Jiarui Wang, Chonghan Liu, and Xu Yang. d^2Cache: Accelerating Diffusion-Based LLMs via Dual Adaptive Caching. *arXiv preprint arXiv:2509.23094*, 2025.

Tomas Kalibera and Richard Jones. Rigorous Benchmarking in Reasonable Time. In *Proceedings of the 2013 International Symposium on Memory Management*, pp. 63–74, 2013. doi: 10.1145/2464157.2464160.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models. *arXiv preprint arXiv:2402.07033*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023. doi: 10.1145/3600006.3613165.

Younjoo Lee, Seungkyun Dan, Junghoo Lee, Jaiyoung Park, and Jung Ho Ahn. DyLLM: Efficient Diffusion LLM Inference via Saliency-based Token Selection and Partial Attention. *arXiv preprint arXiv:2603.08026*, 2026.

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. *arXiv preprint arXiv:2006.16668*, 2020.

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, pp. 32819–32848. PMLR, 2024.

Xinyin Ma, Runpeng Yu, Gongfan Fang, and Xinchao Wang. dKV-Cache: The Cache for Diffusion Language Models. *arXiv preprint arXiv:2505.15781*, 2025.

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, pp. 18332–18346. PMLR, 2022.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, et al. MLPerf Inference Benchmark. In *Proceedings of the 47th Annual International Symposium on Computer Architecture*, pp. 446–459, 2020. doi: 10.1109/ISCA45697.2020.00045.

Subham S. 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, pp. 130136–130184, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv preprint arXiv:1701.06538*, 2017.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Re, Ion Stoica, and Ce Zhang. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 31094–31116. PMLR, 2023.

Bing Tian, Haikun Liu, Xiaocheng Zhong, Zhuohui Duan, Zhaokai Luo, Huayi Jin, Zhiyong Wang, and Xiaofei Liao. FlowBlock: Wavefront-Parallel Decoding for Self-Correcting Diffusion Language Models. *arXiv preprint arXiv:2607.17652*, 2026.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An Insightful Visual Performance Model for Multicore Architectures. *Communications of the ACM*, 52(4):65–76, 2009. doi: 10.1145/1498765.1498785.

Chengyue Wu, Hao Zhang, Shuchen Xue, Shizhe Diao, Yonggan Fu, Zhijian Liu, Pavlo Molchanov, Ping Luo, Song Han, and Enze Xie. Fast-dLLM v2: Efficient Block-Diffusion LLM. *arXiv preprint arXiv:2509.26328*, 2025a.

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. *arXiv preprint arXiv:2505.22618*, 2025b.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache. *arXiv preprint arXiv:2401.14361*, 2024.

Jiacheng Ye, Zhihui Xie, Lin Zheng, Jiahui Gao, Zirui Wu, Xin Jiang, Zhenguo Li, and Lingpeng Kong. Dream 7B: Diffusion Large Language Models. *arXiv preprint arXiv:2508.15487*, 2025.

A Measurement Conventions

Following Section 4, Q counts processed positions across forwards, N forward evaluations, C expert calls, B host-to-device weight movement, and T complete request time. The runtime groups assignments by expert before invoking its projection sequence and records one BF16 weight payload per cache miss. Diagnostics reconcile assignment totals with query visits, layer count, and selected experts per token, and reconcile transferred bytes with weight loads.

B Source-linked Record Schema

Timing records are paired by request, cache mode, and family control, with warm-up and diagnostic records retained separately. Each record identifies the checkpoint, generation budget, runtime configuration, and run status. The evidence ledger links source reports to SHA-256 digests.

For the inputs in Table 2, Fast dual on M20/GSM8K scores 1.0 at 64 tokens; the other configurations, including its matched no-cache control, score 0.0. At 512 tokens, the corresponding configurations score 1.0. These request-level scores are indexed by generation budget. Scoring uses numeric exact match for GSM8K and functional execution for HumanEval, applied to the generated prefix through the first end-of-sequence token when present.

M20 execution extension. Table 3 uses the same execution and paired-timing protocol on inputs selected by task-family context requirements before the extension run. GSM8K/738 repeats an input from Table 2; the remaining identifiers denote additional inputs. The extension records execution costs at 64 tokens. Refresh-1 preserves $Q/N/C/B$ and yields paired latency savings of -0.5% to 0.1%. The input manifest and per-repeat counters accompany the source ledger.

Table 3: M20 execution extension at the 64-token budget. Reductions in $Q/C/B/T$ (%) use family-matched controls; T is mean request latency in seconds. NFE gives the control→cache transition. R4 denotes Refresh-4; MATH-500/1714 identifies `intermediate_algebra/1714.json`.

Input	Mode	$Q\downarrow$	$C\downarrow$	$B\downarrow$	NFE	T (s)	$T\downarrow$
GSM8K/738	Fast prefix	52.3	45.8	62.2	29→33	15.6	44.8
	Fast dual	73.6	48.5	68.3	29→36	15.1	46.6
	dKV R4	52.9	49.2	56.3	64→64	34.1	46.5
GSM8K/138	Fast prefix	69.8	57.0	72.7	44→45	20.2	56.3
	Fast dual	83.1	64.5	79.8	44→42	16.9	63.4
	dKV R4	56.1	50.9	57.4	64→64	35.0	48.4
HumanEval/35	Fast prefix	21.9	22.0	23.4	6→6	4.9	20.4
	Fast dual	30.5	25.1	25.7	6→6	4.7	23.0
	dKV R4	56.8	51.3	58.1	64→64	33.8	48.6
MATH-500/1714	Fast prefix	65.4	54.1	68.0	27→27	12.8	53.4
	Fast dual	74.4	57.7	71.9	27→27	12.0	56.2
	dKV R4	51.1	48.6	55.2	64→64	34.9	46.3

C Replay Comparison Scope

The replay screens use archived native-decoding trajectories from the evaluated settings and count events on those trajectories. Live dKV measurements are recorded separately. Route-triggered refresh is compared with a fixed-four-step schedule. Both require two initial full forwards and force refresh for native verification or changed cached tokens. The trigger uses a 0.05 threshold on the previous visible top-eight expert-set displacement and a maximum refresh age of eight steps; on these traces, it leaves no reuse steps.

The residency screen fixes the query schedule, assignments, and expert calls. With 48 slots per layer, refresh-aware retention protects the preceding partial forward's resident hot set during periodic full refreshes and streams other weights. Outside those refreshes it uses the hot-set control. The reported miss increase is relative to the lower miss count of the hit-first and hot-set controls for each trace. The replay outputs are query-visit and cache-miss counts for the specified schedules and residency rules.

D Mathematical Properties of the Cost Ledger

This appendix derives consequences of the execution contract in Section 4. The coverage results condition on a single layer and forward with fixed routes; the residency bounds use the resident-first ordering in Equation 4. The final identities apply to complete measured requests. These statements formalize why query visits, expert calls, transferred bytes, and forward count capture different aspects of execution.

D.1 Expert demand as a coverage function

Fix a layer and forward, and let $\mathcal{V}$ be the candidate query positions. For each $u \in \mathcal{V}$, hold its routed expert set $R(u)$ fixed. For active positions $A \subseteq \mathcal{V}$, define

$$\mathcal{E}(A) = \bigcup_{u \in A} R(u), \qquad F_\omega(A) = \sum_{e \in \mathcal{E}(A)} \omega_e, \qquad \omega_e \geq 0. \tag{11}$$

Unit weights give the expert-call count for this layer and forward. For a fixed resident set $\mathcal{H}$, choosing $\omega_e = w_e \mathbf{1}\{e \notin \mathcal{H}\}$ gives transferred bytes, where w_e is the expert's weight payload. Thus both quantities are weighted coverage functions, with different weights.

Proposition D.1 (Diminishing marginal expert demand)**.** *The function F_ω is monotone and submodular. Specifically, for $A \subseteq A' \subseteq \mathcal{V}$ and $u \notin A'$, its marginal cost satisfies*

$$\begin{aligned} F_\omega(A \cup \{u\}) - F_\omega(A) &= \sum_{e \in R(u) \setminus \mathcal{E}(A)} \omega_e, \\ F_\omega(A \cup \{u\}) - F_\omega(A) &\geq F_\omega(A' \cup \{u\}) - F_\omega(A'). \end{aligned} \tag{12}$$

Proof. Adding u introduces exactly the experts in $R(u) \setminus \mathcal{E}(A)$, proving the first equality and nonnegative marginal cost. Since $\mathcal{E}(A) \subseteq \mathcal{E}(A')$, the new-expert set for A' is contained in that for A. Summing nonnegative weights proves the second inequality, which is the diminishing-returns definition of submodularity. □

An expert call disappears only when all active positions demanding that expert are removed. For example, if every position routes to the same k experts, any nonempty retained subset still demands k calls. A large reduction in query positions can therefore leave expert-call count unchanged even when the retained routes themselves do not change.

Corollary D.2 (Expected reduction under independent query retention)**.** *For a fixed nonempty A, retain each position independently with probability $\eta \in [0, 1]$, producing A_η. Let $d_e = |\{u \in A : e \in R(u)\}|$ for $e \in \mathcal{E}(A)$. Then*

$$\begin{aligned} \mathbb{E}|A_\eta| &= \eta|A|, \\ \mathbb{E}F_\omega(A_\eta) &= \sum_{e \in \mathcal{E}(A)} \omega_e \big[1 - (1-\eta)^{d_e}\big] \geq \eta F_\omega(A). \end{aligned} \tag{13}$$

If $F_\omega(A) > 0$, its expected fractional reduction is at most $1-\eta$, the expected query-position reduction. The inequality is strict when $0 < \eta < 1$ and some expert has $\omega_e > 0$ and $d_e > 1$.

Proof. Expert e is absent exactly when all d_e positions demanding it are removed, an event of probability $(1-\eta)^{d_e}$. Linearity of expectation gives the sum, without requiring independence between different experts' demand events. Since $d_e \geq 1$, $1-(1-\eta)^{d_e} \geq \eta$, with strict inequality under the stated conditions. □

The multiplicity d_e quantifies route redundancy: experts shared by many query positions are more likely to remain demanded after thinning. This is a conditional explanation of the query–expert gap; applying the expectation to a cache policy requires the stated independent-retention assumption. Actual cache policies can also change routes and future residency.

D.2 Capacity bounds on transferred weights

Fix the demanded set $\mathcal{E}$ at one layer and forward, and write $c = |\mathcal{E}|$. Under resident-first execution with $|\mathcal{H}| \leq K$, the miss count and transfer bytes are

$$m = |\mathcal{E} \setminus \mathcal{H}| = c - |\mathcal{E} \cap \mathcal{H}|, \qquad b = \sum_{e \in \mathcal{E} \setminus \mathcal{H}} w_e. \tag{14}$$

Here $K \in \mathbb{Z}_{\geq 0}$ is the capacity in expert slots.

Proposition D.3 (Tight one-forward residency bounds)**.** *Let $w_{(1)} \leq \cdots \leq w_{(c)}$ be the demanded experts' positive payload sizes and $(x)_+ = \max(x, 0)$. Then*

$$(c-K)_+ \leq m \leq c, \qquad \sum_{j=1}^{(c-K)_+} w_{(j)} \leq b \leq \sum_{j=1}^{c} w_{(j)}, \tag{15}$$

where an empty sum is zero. Both endpoints are attainable over resident sets of size at most K. If all payloads equal w, then $b = wm$ and the byte bounds reduce to $w(c-K)_+ \leq b \leq wc$.

Proof. At most $\min(c, K)$ demanded experts can be resident, giving the miss bounds. To minimize transferred bytes, retain the $\min(c, K)$ largest demanded payloads; the remaining payloads are exactly the smallest $(c-K)_+$ entries. Any other selection retains no greater total payload. An empty resident set attains the upper bounds, and retaining the largest demanded payloads attains the lower bounds. □

The bounds apply to a specified forward; attaining every lower bound along a trace additionally depends on the preceding residency transitions. With a common payload size w, summing the exact identity over the trace gives

$$B = w(C - J), \qquad J = \sum_{t=1}^{N} \sum_{\ell} |\mathcal{E}_{\ell,t} \cap \mathcal{H}_{\ell,t}|. \tag{16}$$

The quantity J counts resident expert calls. Two runs can therefore have the same C but different B: expert-call count specifies demand, whereas the intersection with the resident set specifies which weights must move.

D.3 When additional forwards reduce total work

Consider a matched control 0 and cache mode m with positive forward counts. For any positive measured quantity $W \in \{Q, C, B, T\}$, define $\overline{W}_j = W_j / N_j$ for $j \in \{0, m\}$. For $W = T$, this is amortized request time per forward, including the request-level operations inside the timer boundary in Equation 8.

Proposition D.4 (Forward-count compensation criterion)**.** *Let $\gamma = N_m/N_0 - 1$ and $r_W = 1 - \overline{W}_m/\overline{W}_0$. Then*

$$\frac{W_m}{W_0} = (1+\gamma)(1-r_W), \qquad W_m < W_0 \iff r_W > \frac{\gamma}{1+\gamma}. \tag{17}$$

In particular, with more forwards ($\gamma > 0$), the per-forward reduction must exceed the fraction $\gamma/(1+\gamma)$ to reduce total work or total request latency.

Proof. Substitute $W_j = N_j \overline{W}_j$ into the ratio. Because $1 + \gamma = N_m/N_0 > 0$, rearranging $(1 + \gamma)(1 - r_W) < 1$ yields the stated condition. □

For the measured M20/GSM8K dual transition $29 \to 36$, the threshold is $7/36 \approx 19.4\%$. Its per-forward query, expert-call, and transfer reductions of 78.7%, 58.5%, and 74.4% all exceed this threshold, explaining why all three totals fall despite the extra forwards. The identity for T/N concerns measured end-to-end time; attributing that time to kernels or transfers requires separate timing measurements. The criterion applies to each paired execution before averaging ratios as in Equation 9.

D.4 Repeat weighting in paired latency savings

For $R \geq 1$ timing pairs with $T_{0,i} > 0$, write $s_i = 1 - T_{m,i}/T_{0,i}$, $S_m = R^{-1} \sum_i s_i$, and $\overline{T}_j = R^{-1} \sum_i T_{j,i}$. Define the empirical covariance with denominator R by

$$\mathrm{Cov}_R(T_0, s) = \frac{1}{R} \sum_{i=1}^{R} (T_{0,i} - \overline{T}_0)(s_i - S_m). \tag{18}$$

Lemma D.5 (Paired mean versus ratio of means)**.** *The saving formed from mean latencies, $\widetilde{S}_m = 1 - \overline{T}_m/\overline{T}_0$, satisfies*

$$\widetilde{S}_m = \frac{\sum_i T_{0,i} s_i}{\sum_i T_{0,i}} = S_m + \frac{\mathrm{Cov}_R(T_0, s)}{\overline{T}_0}. \tag{19}$$

If $\sigma_0^2 = R^{-1} \sum_i (T_{0,i} - \overline{T}_0)^2$ and $\sigma_s^2 = R^{-1} \sum_i (s_i - S_m)^2$, then

$$|\widetilde{S}_m - S_m| \leq \frac{\sigma_0 \sigma_s}{\overline{T}_0}. \tag{20}$$

Proof. Use $T_{m,i} = T_{0,i}(1 - s_i)$ to obtain the weighted average. Expanding the covariance gives $R^{-1} \sum_i T_{0,i} s_i - \overline{T}_0 S_m$, proving the identity. Cauchy–Schwarz applied to the two centered sequences gives the bound. □

The paired estimator gives each matched repeat equal weight; the ratio of mean latencies weights it by its control duration. They coincide when the empirical covariance is zero, including equal control durations. This distinction motivates computing the paper's paired savings from the original pairs before rounding the displayed mean latencies.

E Design Implications

The execution ledger separates **token work, demanded expert sets, and expert residency**, following sparse routing and expert placement (Fedus et al., 2022; Jiang et al., 2024; Rajbhandari et al., 2022). Query reuse can shrink an expert's assigned positions without removing its invocation; C therefore counts calls rather than arithmetic operations. Transferred bytes additionally depend on which demanded weights are resident.

These distinctions guide mechanism evaluation: refresh rules change query processing, and residency rules change weight-loading events. The replay diagnostics locate these event changes, while the workload-preserving control supplies a timing reference. Policy evaluation then connects the intended event reduction to complete request execution through the joint (Q, N, C, B, T) ledger.

F Additional Execution Regimes

The supplemental experiments examine residency, capacity, hardware, shared dispatch, and input diversity. Each comparison retains its own protocol: trace projections report event counts, while GPU executions also report time. The data manifest links input identities, configurations, and source records.

F.1 Persistent-cache request streams

On A800 with pageable host weights, both checkpoints use 36 inputs from six datasets, 128 slots per layer, and 16 generated tokens per request. Demand-LRU is compared with protected-hotset admission, which retains experts by current-layer routing frequency. One FIFO worker processes six requests per stream, preserving the cache between requests. Four arrival-order seeds and two balanced repeats per policy give 96 streams per checkpoint. Routed assignments and expert calls remain fixed; both policies reproduce the full-resident reference's generated tokens and ordered routes.

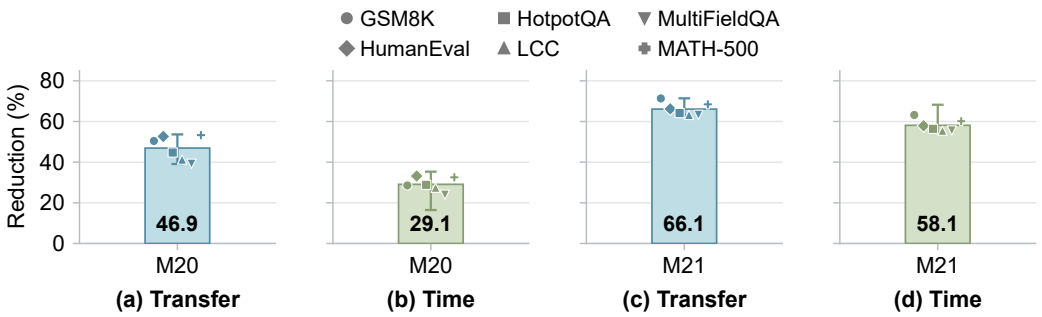

Figure 6: Persistent-cache streams: transfer and synchronized stream-time reductions for M20 (a)–(b) and M21 (c)–(d). Bars average six dataset means; markers identify datasets, and capped lines span their arrival-order reductions. Comparisons use identical inputs and 128 slots per layer.

The stream timer spans the first scheduled arrival to final synchronized completion, including tokenization and waiting. Repeats are averaged within each arrival order before computing the relative reduction; the four ratios are then averaged. Table 4 weights datasets equally. Transfer reductions of 46.9% and 66.1% accompany stream-time reductions of 29.1% and 58.1% on M20 and M21, respectively, at unchanged routed workload and C.

Table 4: Persistent-cache streams: reductions (%) relative to Demand-LRU at 128 slots per layer. Each cell averages paired arrival-order results; the final row gives equal weight to the six datasets.

	LLaDA2.0-mini		**LLaDA2.1-mini**	
Dataset	$B \downarrow$	$T_{\text{stream}} \downarrow$	$B \downarrow$	$T_{\text{stream}} \downarrow$
GSM8K	50.4	28.5	71.3	63.2
HumanEval	52.6	33.2	66.2	57.9
HotpotQA	44.7	28.8	64.1	56.4
LCC	41.3	27.4	63.2	55.5
MultiFieldQA	39.2	24.0	63.3	55.6
MATH-500	53.3	32.5	68.4	60.1
Macro mean	**46.9**	**29.1**	**66.1**	**58.1**

F.2 Capacity sensitivity at fixed expert demand

The sweep replays six 32-token M21 trajectories, preserving 817,152 assignments and 125,187 expert calls across five capacities. Protected admission retains experts by current routing frequency; resident-first serves resident experts before misses, and denoise-survivor additionally uses preceding denoising-step membership. Each policy uses a matched-capacity protected-admission control.

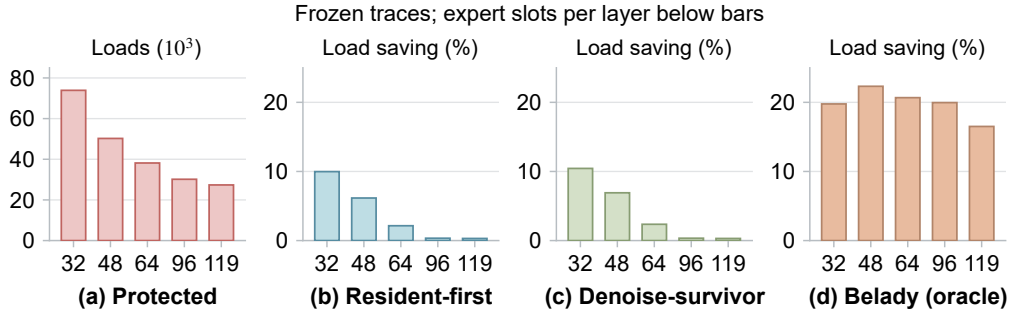

Figure 7: Capacity sensitivity on frozen traces. (a) Total protected-admission loads across six traces. (b)–(d) Load reductions relative to protected admission at each matched capacity. Tick labels give expert slots per layer. Each load carries 6 MiB; Belady uses future accesses. Source generation uses 119 slots; all panels report replay counts.

Increasing protected-admission capacity from 32 to 64 slots reduces loads from 73,900 to 38,164 (48.4%) at unchanged C; 119 slots yield 27,365 loads. The resident-first saving changes from 10.0% at 32 slots to 2.1% at 64 slots. Capacity affects both transfer demand and relative policy differences.

F.3 Transfer and latency on RTX 3090

RTX 3090 execution uses pinned host weights, the same six M21 inputs, 32 BF16 slots per layer, and 32 generated tokens. Each policy has four primary repeats with reversed order and separate diagnostics. Tokens and ordered routes match a streamed BF16 reference. Primary timers include route capture and slot snapshots.

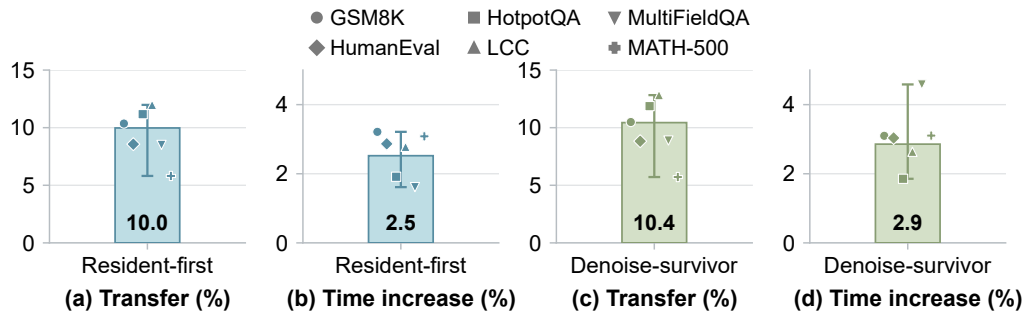

Figure 8: RTX 3090 measurements relative to the equal-capacity BF16 baseline: resident-first (a)–(b) and denoise-survivor (c)–(d). Bars use ratios of summed primary measurements; markers give per-input ratios, and capped lines span inputs. Transfer reductions and time increases are percentages, with four primary repeats per policy and input.

Summing primary records before forming ratios, resident-first and denoise-survivor reduce B by 10.0% and 10.4%, while time increases by 2.5% and 2.9%. Figure 8 resolves the changes by input. These measurements distinguish avoided weight movement from request latency: the reduction in B alone does not establish a reduction in T.

F.4 EXPERT ACCOUNTING IN PACKED COHORTS

Packed execution changes the unit at which expert demand is combined. The A800 experiment concatenates independent M21 requests with block-diagonal attention and shared expert dispatch, using pinned host weights, 128 slots per layer, block length 16, and fixed four-token commitment. Table 5 reports the confidence-first baseline for every homogeneous cohort of size 1, 2, or 6, covering 36 input identities. Each execution is checked against its full-resident reference. The cohort timer includes packing, route capture, and synchronized generation; serialization follows timing.

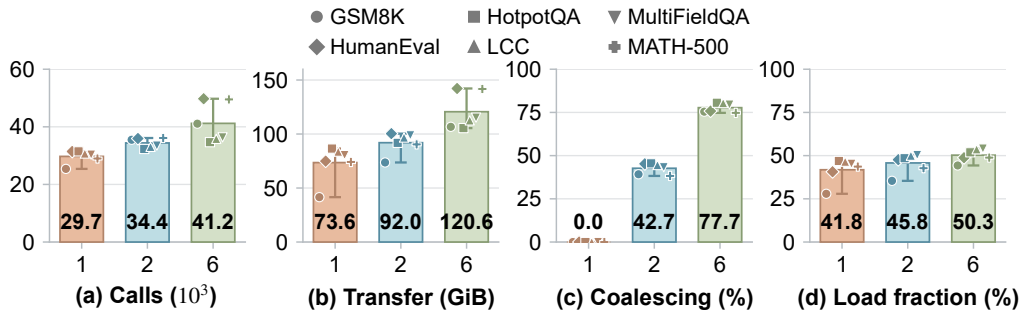

Figure 9: Packed-cohort accounting: (a) expert calls C, (b) transferred bytes B, (c) coalescing $100(1 - C/C_{\text{separate}})$, and (d) load fraction $100B/(wC)$. Here C_{separate} sums request-specific expert groups on the same routes and $w = 6$ MiB. Tick labels give requests per cohort. Bars average six datasets, markers identify their measured configurations, and capped lines show min–max ranges.

Six-request cohorts coalesce 74.7–80.6% of the request-specific expert groups, while 44.3–54.1% of their shared calls require a weight load. These are separate accounting effects: grouping changes C, and residency determines the load fraction $B/(wC)$. Cohort sizes use their frozen input memberships; Table 5 reports the corresponding measured $C/B/T$ costs.

Table 5: Measured packed-cohort costs under the confidence-first baseline: expert calls C (10^3), transferred bytes B (GiB), and mean cohort time T (s) across the two primary repeats. Each request generates 32 tokens.

	1 request			2 requests			6 requests		
Dataset	C	B	T	C	B	T	C	B	T
GSM8K	25.4	41.5	7.1	35.5	73.7	9.9	41.1	106.7	12.1
HumanEval	31.4	74.9	8.9	35.9	100.4	10.4	49.7	142.2	15.3
HotpotQA	31.5	86.5	9.6	32.3	91.8	10.3	34.7	105.5	14.2
LCC	30.7	83.7	9.3	33.0	96.7	10.4	35.8	112.6	14.7
MultiFieldQA	30.3	80.5	9.5	33.5	98.8	10.6	36.3	115.2	14.7
MATH-500	29.0	74.2	8.4	36.1	90.4	10.6	49.5	141.7	15.1

F.5 Input diversity in recorded expert demand

The broader archive contains 108 M20 inputs across nine datasets and 972 policy executions. We analyze the common four-token-commit baseline for all inputs, combining the archived discovery and holdout partitions descriptively. Generation uses 16 output tokens; Table 6 lists the recorded prompt lengths and forward counts.

The archive records expert sets for the whole 16-position block and its active positions at each forward. For each region, we sum position counts (Q_v) across forwards and expert-set cardinalities (C_v) across layers and forwards. Figure 10 reports $100(1 - Q_{\text{active}}/Q_{\text{block}})$ and $100(1 - C_{\text{active}}/C_{\text{block}})$ per input. This fixed-route projection measures changes in position and expert-group counts within the block.

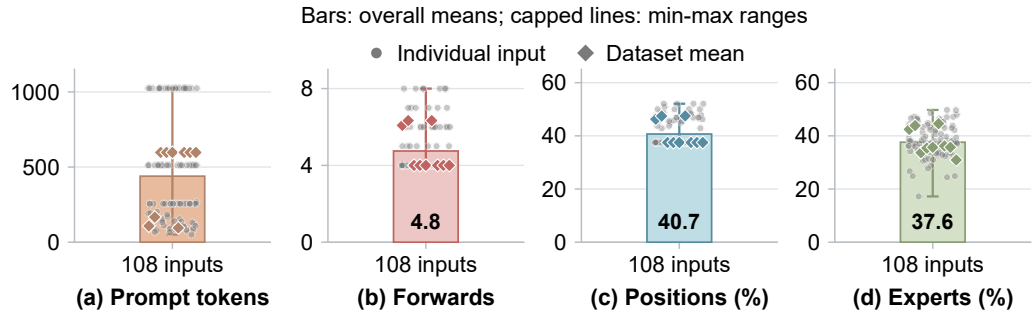

Figure 10: Recorded inputs and fixed-route projections. (a)–(b) Prompt lengths and forward counts. (c)–(d) Position and expert-group reductions on the same trajectories, with a shared percentage scale. Bars average all 108 inputs; gray points retain individual inputs, diamonds show dataset means, and capped lines show input min–max ranges.

The dataset means distinguish position reduction from expert-group reduction across all nine tasks. The six LongBench tasks each have 37.5% mean position reduction, yet their expert-group reductions range from 30.9% to 36.2%. Thus, the same fraction of omitted positions corresponds to different retained expert sets. Overall means are 40.7% for positions and 37.6% for expert groups.

Table 6: Recorded input lengths and active-position projection on the common four-token-commit baseline. Lengths and NFE are min–max ranges; reductions are mean per-input ratios (%) relative to the recorded whole-block demand. Each dataset contributes 12 inputs.

Dataset	Prompt tokens	NFE	Positions↓	Expert groups↓
GSM8K	75–156	4–8	46.3	42.4
HumanEval	91–267	4–8	47.4	43.9
HotpotQA	256–1024	4	37.5	33.7
LCC	256–1024	4	37.5	35.4
MultiFieldQA	256–1024	4	37.5	35.7
MATH-500	51–153	5–8	47.5	44.6
PassageCount	256–1024	4	37.5	36.2
SAMSum	256–1024	4	37.5	35.7
TREC	256–1024	4	37.5	30.9
Macro mean	–	–	**40.7**	**37.6**