FRAME: ROUTING-AWARE EXPERT PLACEMENT FOR CPU–CXL MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-experts (MoE) models activate few experts per token, yet serving requires access to the full expert set, so expert-weight memory requirements scale with total rather than active parameters. Keeping all weights in GPU memory is costly, while offloading can stall decoding with weight transfers. CPU servers with DDR5 and CXL memory expand available capacity and allow CPU cores to execute experts directly from either tier. Efficient placement remains challenging: DDR5 capacity is limited, expert activation patterns change across requests, and migration into DDR5 competes with in-place expert reads for CXL bandwidth. We present FRAME, a routing-aware expert-placement system for CPU–CXL MoE inference. FRAME separates expert ranking from DDR5 admission so that changing routing estimates need not trigger immediate migrations. An online adaptive mechanism estimates residency benefits from recent expert activations, while a residency controller selects between frequency-based and adaptive rankings according to how well historical frequencies capture current routing patterns. Bounded admission and eviction regulate the resulting migrations, enabling placement adaptation while controlling its CXL traffic cost. We evaluate FRAME on a real CPU–CXL system across three MoE architectures and controlled and held-out workloads. Following a routing-distribution shift introduced without notifying the runtime, FRAME achieves 17.09 tokens/s on a cross-domain request sequence, outperforming the strongest adapted baseline by 17.6%. It also achieves 16.92 tokens/s on 24 held-out LongBench requests spanning six tasks. Mechanism analysis shows how ranking and admission jointly determine the trade-off between reducing subsequent CXL reads and incurring migration traffic.

1 INTRODUCTION

Mixture-of-experts (MoE) models increase model capacity while activating only a small subset of experts for each token (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). This computational sparsity does not eliminate the memory requirement: different tokens and requests can select different experts, requiring access to a much larger set of weights than any individual token uses. Keeping these weights in GPU memory becomes increasingly costly as models grow, while fetching missing weights can stall decoding. These constraints motivate efficient MoE inference on CPU servers without GPUs. Such servers can combine local DDR5 with CXL-attached memory to accommodate large models, but the bandwidth disparity between these tiers makes expert placement critical to decoding performance.

Existing MoE offloading systems relieve accelerator-memory pressure by storing expert weights in host DRAM, using caching to reduce repeated transfers and prefetching to overlap transfers with computation (Eliseev & Mazur, 2023; Xue et al., 2025). SSD backing further extends capacity beyond DRAM, but accessing weights absent from memory requires storage I/O before execution. In our CPU-only setting, CXL memory provides another option: CPU cores can access CXL-resident weights through load instructions and execute the corresponding experts without first copying them into DDR5. An expert missing from DDR5 therefore remains directly executable, although at lower memory bandwidth. This gives the runtime a choice between executing an expert from CXL and migrating its weights into DDR5 for faster subsequent accesses.

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 illustrates the performance consequences of these choices on our CPU server. With 10 GiB of DDR5 reserved for expert weights, mmap-based SSD backing achieves 4.49 tokens/s, with demand-driven storage reads delaying decoding. Placing all expert weights in CXL memory avoids these storage reads and increases throughput to 10.70 tokens/s. Keeping all experts in DDR5 reaches 19.40 tokens/s, but requires enough local memory to hold the entire expert set. These results expose the capacity–performance trade-off: CXL accommodates experts beyond the DDR5 budget, while DDR5 residency offers substantially faster execution. Exploiting both tiers requires allocating the limited DDR5 capacity to experts whose future accesses benefit most from that placement.

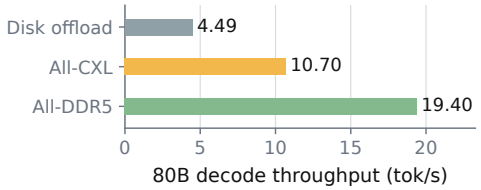

Figure 1: Measured 80B decode throughput

Two coupled challenges govern this allocation. First, expert demand is skewed and varies across requests, so placements based on historical activation frequencies can become stale as the request mix changes. Activation tracing and prediction help anticipate this demand (Xue et al., 2025), but accurate demand estimates alone do not determine whether migration is worthwhile. This leads to the second challenge: migration consumes the same CXL bandwidth needed to read weights for in-place execution. Its cost is incurred immediately, whereas its benefit depends on the accesses served from DDR5 before the expert is evicted. Frequent placement changes can therefore consume bandwidth without sufficient reuse to recover their cost, while infrequent changes leave DDR5 occupied by experts that no longer match current demand. Effective expert placement must jointly adapt to routing changes and control the migration traffic required to maintain DDR5 residency.

To address these challenges, we present FRAME, a routing-aware expert-placement system for CPU–CXL MoE inference. Its central design separates expert ranking from DDR5 admission, allowing rankings to adapt without triggering migration on every change. An online adaptive mechanism estimates residency benefits from recent expert activations, while a residency controller selects between frequency-based and adaptive rankings according to how well historical frequencies represent recent routing patterns. This selection preserves stable placement when activation frequencies remain consistent and adjusts expert priorities when routing patterns shift. The controller then bounds admission and eviction, limiting the migration traffic required to update DDR5 residency and its contention with in-place CXL reads.

We evaluate FRAME on a real CPU–CXL system across three MoE architectures and controlled and held-out workloads. A cross-domain sequence from GSM8K, MMLU, and WMT16 introduces a routing-distribution shift without revealing the task boundary to the runtime. Under stable routing, FRAME uses frequency-based ranking and outperforms prediction-based baselines by 2.0–2.6 tokens/s. After the shift, it switches to adaptive ranking and sustains 17.09 tokens/s, exceeding the strongest adapted baseline by 17.6%. It also achieves 16.92 tokens/s on 24 held-out LongBench requests spanning six tasks. Mechanism analysis connects ranking and admission decisions to migration traffic and subsequent CXL reads, explaining when adaptation yields gains sufficient to offset its transfer cost.

We make the following contributions:

(1) **Characterizing the execution–migration trade-off.** We characterize MoE decoding on a real CPU–CXL system and quantify the performance gap between direct CXL execution and DDR5 residency. We show how expert migration competes with in-place reads for CXL bandwidth, making migration cost a key consideration beyond expert activation frequency.

(2) **Adaptive expert placement with controlled migration.** We design FRAME, which separates expert ranking from DDR5 admission. A routing-aware controller selects between frequency-based and online adaptive rankings, while bounded admission and eviction control the migration traffic induced by placement changes. Together, these mechanisms adapt expert residency to changing routing patterns while limiting interference with in-place execution.

(3) **System evaluation and mechanism analysis.** We evaluate FRAME across three MoE architectures, controlled routing shifts, and held-out requests. Beyond measuring decode throughput, we analyze how ranking and admission decisions affect migration traffic and subsequent CXL reads, explaining when placement adaptation delivers sufficient reuse to offset its migration cost.

2 Related Work

MoE models activate a sparse subset of experts per token, producing an input-dependent working set whose reuse can vary across requests (Shazeer et al., 2017; Fedus et al., 2022; Jiang et al., 2024). Existing systems exploit this sparsity through expert caching, prediction, heterogeneous execution, and compression, but the placement objective depends on the underlying memory hierarchy.

Expert offloading and caching. Existing systems manage expert transfers by loading missing experts before execution. Demand paging (Eliseev & Mazur, 2023), FineMoE (Yu et al., 2025), FluxMoE (Liu et al., 2026) and ZipMoE (Yang et al., 2026) reduce or schedule expert movement under this execution model. However, these approaches assume that a nonresident expert must be transferred before execution. That assumption no longer holds in a CPU–CXL system: the expert remains executable in place, which makes migration a placement optimization rather than a prerequisite.

Predictive and workload-aware expert management. Another line predicts future expert demand to guide caching or prefetching. MoE-Infinity (Xue et al., 2025), ProMoE (Song et al., 2025), ExpertFlow (He et al., 2026) and Fate (Fang et al., 2025) estimate future expert requests from activation patterns, routing correlations or cross-layer gates. Prediction remains valuable for identifying future residency opportunities, but its benefit depends on whether the resulting residency changes reduce more CXL demand traffic than they introduce.

Heterogeneous execution and CXL memory. Another line explores heterogeneous execution and memory expansion, in three groups: computation placement between CPU and GPU (Kamahori et al., 2025; Chen et al., 2025; Zhu et al., 2026; Wang et al., 2026), parameter representation and locality (Song et al., 2023; Tang et al., 2024), and CXL as a capacity tier for pooled or remote memory (Li et al., 2023; Yang et al., 2025; Yoon et al., 2025; Kim et al., 2024). These works differ from our setting: we fix CPU execution and model representation, and study how expert residency should be managed when CXL already provides executable access to nonresident experts.

Positioning. Existing systems primarily optimize expert transfer, heterogeneous execution, or CXL capacity expansion. FRAME instead studies expert residency control in a CPU–CXL MoE system, where migration is optional and must justify the bandwidth it consumes. Prediction remains an important signal in this setting, but effective placement requires coupling prediction with migration-cost-aware residency control.

3 Methodology

FRAME combines an online adaptive expert ranking mechanism with a routing-aware placement controller. The ranking mechanism answers which experts become important, the selector answers when adaptation should be trusted, and the residency controller answers whether moving them is worth the bandwidth. Fig. 2 shows the execution path and the control loop in one place, so that the tier a layer reads from and the request-level decision that put the expert there can be read together. We define the residency objective and its metrics, then describe the predictor and the controller that produce the deployed policy.

3.1 Problem formulation

For request r, token t, and MoE layer l, let $A_{r,t,l}$ be the routed expert set, $k_l = |A_{r,t,l}|$, and P_l the candidate capacity. Given each layer's routing history, FRAME maintains an ordered expert ranking that estimates residency benefit: the first P_l unique entries of that ranking form $S_{r,t,l}$. The ranking must be issued before the target route becomes available, and may use previous-token routes and other signals only when their observation timestamps precede the issue time, so ranking accuracy alone does not capture system benefit. The ranking does not trigger migration by itself – it is the input to the placement decision below.

Let $R_{r,t,l}$ denote the DDR5-resident experts that are valid and usable when the layer executes. In an asynchronous runtime S and R differ: some proposals are rejected by admission control, some

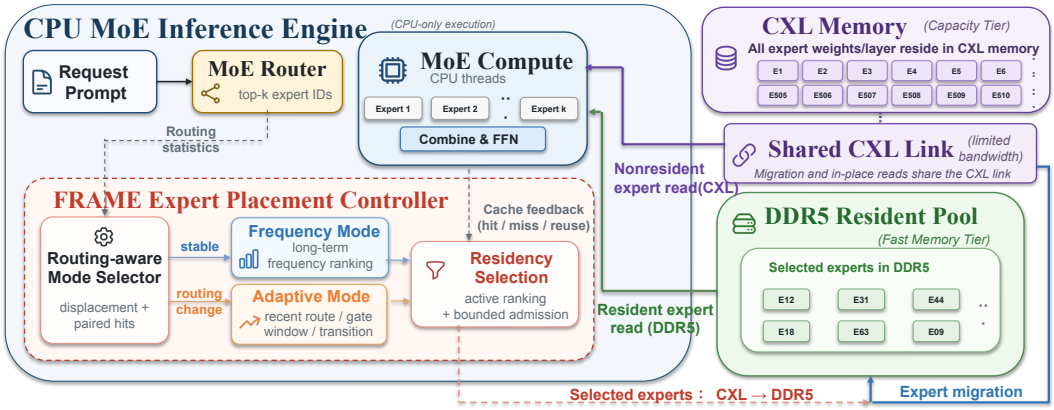

Figure 2: FRAME on a CPU–CXL host. All expert weights live in the CXL capacity tier; the controller chooses which experts are resident in DDR5; the CPU executes a layer from DDR5 when its expert is resident there, and reads it in place from CXL otherwise. In-place reads and the migrations filling the DDR5 resident pool share one limited CXL link. Dashed arrows are feedback.

copies arrive late, some become stale. The physical constraint is on the resident payload,

$$\sum_{l} \sum_{e \in R_{r,t,l}} b_{l,e} \le B_{\text{pool}}, \tag{1}$$

where $b_{l,e}$ is the stored byte size of expert e in layer l and B_{pool} is the routed-expert DDR5 pool budget, which excludes other model weights, activations, runtime state and KV cache. The bound is on the resident payload: the engine allocates with its own eviction hysteresis and accounts for in-flight copies separately, so peak allocation can briefly exceed the payload accounted here (Appendix A.3).

The predictor is evaluated by the coverage of its issued candidates:

$$\text{Recall@}P = \frac{\sum_{r,t,l} |A_{r,t,l} \cap S_{r,t,l}|}{\sum_{r,t,l} k_l}. \tag{2}$$

Its *candidate turnover* $C = |S_{r,t,l} \setminus S_{r,t-1,l}|$ counts newly proposed experts per decode-layer event. Recall alone cannot represent end-to-end benefit, because realizing a ranking costs migration traffic that competes with the in-place CXL reads it is meant to avoid; the metrics below therefore pair it with what the runtime realizes. *Ready recall* uses R instead of S in Eq. (2), measuring whether the required DDR5 copy exists when computation reaches the layer. We report *completed migration bytes* B as the copy volume of one run. Admitting one expert copies all three of its weight tensors—its gate, up and down projections, and the migration volume sums all three, although only the gate and up copies respond to the policy (Appendix D.1). The issued volume V counts what the engine submits for migration in the same run. For replay analysis, *logical churn* M charges each newly admitted expert its stored byte size; it is distinct from measured device traffic. Appendix E.1 defines replay-resident recall and the two volume quantities.

The system objective is decode throughput under Eq. (1). At predictor level we compare the recall–turnover frontier at a target recall ρ:

$$\text{minimize } C \quad \text{subject to} \quad \text{Recall@}P \ge \rho. \tag{3}$$

The live engine then measures how the resulting candidates affect ready recall, migration volume, and tokens/s, and where instrumented, direct CXL traffic.

These metrics separate ranking quality (candidate recall and turnover C), realized residency (ready recall with the issued and completed volumes V and B), and platform impact (logical churn M, direct CXL traffic, tokens/s), so a predictor-level gain is not read as a system gain.

3.2 Online Adaptive Residency Ranking

Long-term frequency provides an effective residency signal under stable routing, but it cannot react quickly when expert demand changes across requests. FRAME therefore introduces an online

adaptive expert predictor that complements the frequency signal by continually estimating expert relevance from recent routing observations. The predictor is a residency-ranking component rather than an independent prefetch mechanism: it forecasts no execution demand and issues no transfer itself, and its objective is to hand the placement controller better candidates as the routing changes. It combines signals that capture complementary temporal and routing characteristics of expert activation: window frequency, route state, first-order transition, and gate logits where available (Appendix E.1). Each yields a per-layer expert ranking, and the predictor weights them by their realized hit rate. Lifetime frequency is scored causally too, but its measured share sets the ranking-anchor width F_{rank} rather than entering the blend (Section 3.4).

For expert e in layer l, the adaptive score is

$$q_l(e) = \sum_{j \in \mathcal{J}} w_{j,l}\, q_{j,l}(e), \qquad \sum_{j \in \mathcal{J}} w_{j,l} = 1, \tag{4}$$

with $\mathcal{J}$ the four dynamic signals above, $q_{j,l}$ a normalized signal ranking and $w_{j,l}$ following that signal's measured hit rate.

The predictor is not fitted offline: it scores each signal against the routes realized after that signal's proposal was issued and updates its weights from the feedback,

$$\rho_j \leftarrow \gamma\, \rho_j + (1-\gamma)\, \mathrm{hit}_j, \qquad w_{j,l} = \frac{\bar{\rho}_j + b}{\sum_{j' \in \mathcal{J}} (\bar{\rho}_{j'} + b)}, \tag{5}$$

with hit_j the share of the realized route that signal j's proposal covers, $\bar{\rho}_j$ the running mean of ρ_j pooled over layers, γ the update rate and b a weight floor that keeps a signal which loses one request from being dropped; the numerical values, the scoring cadence and the fallback before the first reward are in Table 16. Online feedback is designed to regulate how aggressively the ranking may be acted on rather than to maximize predictor accuracy on its own. The predictor therefore adapts inside the serving process, with no offline refit, no held-out tuning set and no restart when the routing changes. The same running estimate sets the admission cap of Section 3.4: the predictor governs what is ranked, the controller governs what is admitted.

The deployed predictor costs 21–99 μs per call across the three evaluated geometries, against 1.29–2.24 ms for one expert migration.

3.3 Routing-aware selection

Unlike the MoE router, which chooses which experts compute a token, FRAME's selector chooses the ranking strategy used for residency management: it picks between a frequency ranking and the predictor's causal ranking for the same DDR5 pool. At the end of each request's prompt, before generation begins, it counts the experts routed in that prompt. For each layer, let $T_{r,l}$ be the most frequent experts observed in request r, truncated to the number observed or the pool capacity, whichever is smaller, and let F_l be an equally wide prefix of the current frequency ranking. The request-level shift score is

$$d_r = \mathrm{median}_l \left(1 - \frac{|T_{r,l} \cap F_l|}{|T_{r,l} \cup F_l|} \right). \tag{6}$$

A request with insufficient prompt evidence leaves the mode unchanged. In **frequency** mode, d_r above the shift threshold (Table 16) makes the adaptive ranking *eligible* for the whole pool. Crossing the threshold is an eligibility test rather than a cost calculation: the mode changes only if the adaptive ranking's hit count over the request's own prompt routes exceeds the frequency ranking's, and the normalized hit advantage $(H_{\mathrm{ad}} - H_{\mathrm{fr}})/\max(1, N_{\mathrm{ad}} - N_{\mathrm{fr}})$ clears the increment margin (Table 16; the deployed margin is 0, and Appendix A.4 calibrates the band), where N_{ad} and N_{fr} count the experts each ranking proposes that were not resident before the request. **Frequency** mode exploits long-term expert reuse under stable routing; **adaptive** mode refines that ranking when recent routing evidence indicates that the historical statistics have become stale.

While in **adaptive** mode, the controller accumulates decode routes into a candidate frequency ranking for the new distribution and compares its hit count with the adaptive ranking's on prompt observations where both were issued before the route was known. A median per-layer cosine distance between the candidate and the current prompt detects a further routing change and resets the candidate; otherwise two consecutive requests with valid paired evidence and candidate hits at least as

high as the adaptive ranking's pin it and return to **frequency**. Missing paired evidence keeps the adaptive mode, every decision uses only routes observed before the generation it controls, and the shared capacity, bounded admission rule and engine-side eviction hysteresis below limit the migration traffic either mode creates.

3.4 RESIDENCY CONTROL

Although the predictor improves ranking quality, a migration decision must also weigh the CXL bandwidth that expert reads and expert movement share: the predictor affects what is ranked, the admission controller below whether the resulting change justifies its migration cost. FRAME splits a layer's pool into a frequency-preserved region of F_{eng} slots, holding the current top-frequency experts, and an adaptive replacement region of D_{eng} slots that the policy ranks. Both blocks exist in either state, but the tail is dynamic only in **adaptive** mode – in **frequency** mode the policy's slots follow the frequency ranking as well, so the pool is one frequency-determined set. The core is a ranking priority rather than a pinned set: the top-F_{eng} experts are recomputed from the online estimate at every issue call and appended after the policy's D_{eng}, so the two blocks compete for the same slots, and the policy-side retention bonus is 0 (Appendix E.2). The predictor-side anchor F_{rank} protects the highest-frequency experts while the multi-signal ranking fills the other $P_l - F_{\mathrm{rank}}$ candidate slots; it is not tuned but implied by the online frequency weight (αP_l, about 5 slots on the 80B), and every reported system run holds it there – removing it changes the ranking's recall and turnover in replay, which is a ranking-level measurement (Appendix C.1). The two widths are therefore measured at different levels, and no reported run varies both at once.

The split is what the adaptive state runs on: the core buys migration reduction, the dynamic tail buys adaptation, and enlarging the tail is not free. Expert migration and in-place reads share the CXL path, so the split trades migration traffic against the in-place reads it delays; it is sized by one cost law. For a split F,

$$\frac{1}{\mathrm{tok/s}} = a + b\,M(F), \qquad \mathcal{F} = \big\{F :\ \hat{t}(F) \geq \hat{t}^{*} - \delta\big\}, \tag{7}$$

with $M(F)$ the logical churn, $\hat{t}(F)$ the throughput it predicts, a fixed per-token time, b the marginal logical-churn cost and $\delta = 0.1$ token/s. The coefficients are fitted on the calibration prompts and then frozen, and the frozen law is applied blind to a disjoint validation pass, where it retains an interval of splits rather than a single value: the split the deployed system uses lies inside that interval, and the applied widths, the fit and the sweep that checks them are reported with the deployment rule and in Appendices D.2 and D.3.

An admission budget bounds the replacements introduced at each opportunity instead of forcing a refill after every prediction. We set

$$a_{t,l} \approx P_l\big(1 - \widehat{h}_{t,l}\big), \tag{8}$$

rounded and clipped, where $\widehat{h}_{t,l}$ is a causal running estimate of the fraction of issued predictions that were correct: a high hit rate suppresses migration traffic, a lower one lets the adaptive ranking replace more of the resident set.

Execution contract and deployment. Routed expert weights are backed by CXL memory and the experts selected for residency are copied into a bounded DDR5 pool: a valid resident copy is read from DDR5, and otherwise the host executes directly from the CXL-backed address, so a missing expert costs latency rather than correctness. A migration publishes its DDR5 location only after the copy completes, and eviction invalidates the read-only copy, so a partially copied tensor is never read. The method changes neither the router's selection nor the precision of the quantized reference.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

We evaluate Qwen3-Next-80B-A3B-Instruct (Q4_K_M) on a two-socket Intel Xeon Gold 5520+ server (112 threads) with 64 GB DDR5-4800 modules per socket and a CXL 2.x-attached 258 GiB

memory node as the second tier, whose in-place expert-sized read measures 14.83 GB/s (Appendix A.2); the engine uses 24 threads and a 10 GiB DDR5 expert pool. The main experiment is one continuous 14-request stream (six GSM8K, then four MMLU and four WMT16), with all policy arms inside a comparison campaign sharing one engine binary, model, CPU/NUMA binding, copy path and 32-token generation limit for every policy, the pool mutable in every arm (FRAME's evaluation-time residency freeze after 30 issue calls is stated and ablated in Appendix A.3; the adapted baselines freeze nothing), and prefill excluded from throughput. Bindings and the 120-token LongBench campaign are in Appendix B.4.

Baselines.

We compare FRAME with three classes of expert-management policies: static placement and conventional cache replacement (Static-Freq and Uniform (Chen et al., 2025), LRU (Eliseev & Mazur, 2023)), reuse-aware admission (Cost-Gated (Kamahori et al., 2025)), and predictive expert prefetching or caching (MoE-Infinity (Xue et al., 2025), ProMoE (Song et al., 2025), Pre-Gated MoE (Hwang et al., 2024), ExpertFlow (He et al., 2026)). Each arm re-implements the mechanism of the cited paper in the same CPU–CXL runtime; Appendix A.1 documents the mapping. All-CXL and All-DDR5 are the memory-tier references, and fixed frequency and fixed adaptive isolate the two FRAME rankings.

Residency configuration.

The 80B pool holds $P = 107$ slots per layer (30B $P = 29$, OLMoE $P = 19$); Section 5.2 and Appendices D.2 and D.3 analyze its operating region.

Workloads. The regime stream is 14 requests (six GSM8K, four MMLU, four WMT16) decoded under a **32-token** cap. The held-out expansion repeats the comparison on 24 LongBench requests from six tasks, disjoint from the stream, under the **120-token** cap of Appendix B.4.

4.2 System-level decode performance

The predictor is not designed to maximise recall alone: it supplies the residency controller with candidates whose turnover the CXL link can afford, so its ranking is read at matched turnover.

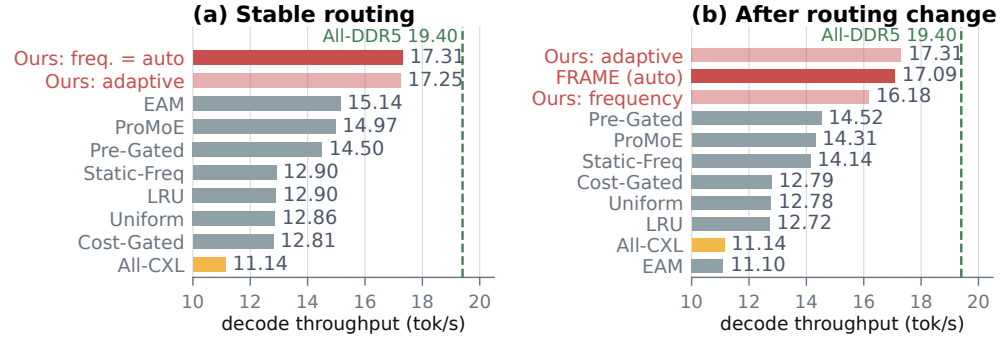

Figure 3: System-level decode throughput on the controlled 14-request stream (10 GiB pool, 32-token cap). (a) Six stable-routing GSM8K requests; (b) eight MMLU/WMT16 requests after the routing change. Axes begin at 10.

Fig. 3 shows how FRAME selects a ranking as the stream changes. On stable routing it holds the frequency ranking, so panel (a) draws auto and fixed frequency as one merged row taken from the frequency arm (17.31 tokens/s, one pass) while fixed adaptive is 0.12 tokens/s lower across the three replicate passes (17.27 against 17.39) and moves 13.9% more data. It switches at the first MMLU request and reaches 17.09 tokens/s over the eight post-transition requests: 5.6% above fixed frequency (16.18) and 17.6% above the strongest adapted baseline, Pre-Gated (14.52). The selector's own log makes that choice explicit: frequency for the six stable requests, adaptive for all eight post-transition ones. It therefore lands on the better fixed policy in each segment — identical to the frequency policy on the stable segment (23.79 against 23.81 GiB of completed migration at the same throughput) and, over the whole stream, within 0.3% of fixed adaptive at 7.0% less completed migration (paired $+0.004$ tokens/s, 95% interval $[-0.035, +0.043]$ over 50 request pairs) — whereas a fixed choice pays one of those costs: always-frequency loses 5.6% after the shift, and

always-adaptive moves 13.9% more data on the stable segment. Over the whole stream FRAME is 2.6% ahead of always-frequency. Three further passes reproduce the ordering (Appendix B.3).

Held-out workload expansion. The cross-domain sequence is our own construction, so we repeat the comparison on the 24 held-out LongBench requests of Section 4.1, same engine and pool: FRAME delivers 16.92 tokens/s against Pre-Gated (14.38), Static-Freq (13.98) and All-CXL (10.67), $+2.54$ tokens/s ($+17.7\%$) over the strongest adapted baseline and ahead on all six tasks (Appendix B.4).

4.3 Cross-model results

We next evaluate FRAME on Qwen3-30B-A3B (4 GiB pool, $P = 29$) and OLMoE-1B-7B (1 GiB, $P = 19$); all three geometries allocate 22.2–25.5% of the model file to the expert pool, so the budgets are approximately matched as a fraction of model size; the online policy constants are shared across all three (Table 16) and only P and F_{eng} are per-geometry (Appendix D.3).

On the 30B the profiled Static-Freq baseline is favorable on the GSM8K prefix used to build its hot set, so the full-stream means nearly tie (21.22 against 21.19 tokens/s); once the workload leaves that profile FRAME sustains 21.29 tokens/s on MMLU–WMT16 against 17.98, a $+18.4\%$ gain. A matched state control at fixed residency settings isolates the state: forcing frequency is 1.91 tokens/s slower than auto and forcing adaptive reproduces auto (Appendix B.5). OLMoE supplies a third, smaller geometry: FRAME reaches 38.33 tokens/s against 32.92 for LRU ($+16.4\%$), so the same residency controller remains effective across model geometries.

Table 1: Cross-model system behavior against the strongest adapted baseline for each geometry. "Region" denotes the evaluated portion of each pass.

Model	Region	Pool	FRAME	All-DDR5	Strongest adaptation	Gain
Qwen3-Next-80B	post-shift	10 GiB	**17.09**	19.40	Pre-Gated (14.52)	**+17.6%**
Qwen3-30B	MMLU–WMT16	4 GiB	**21.29**	34.87	Static-Freq (17.98)	**+18.4%**
Qwen3-30B	full pass	4 GiB	**21.22**	34.87	Static-Freq (21.19)	$+0.1\%$
OLMoE-1B-7B	full pass	1 GiB	**38.33**	58.80	LRU (32.92)	**+16.4%**

5 Analysis

5.1 Ranking effectiveness

Table 2 compares causal rankings at $P = 107$ on 16 held-out Qwen3-Next requests, testing both sides of the trade: FRAME recall within each baseline's turnover budget and FRAME turnover at each baseline's recall.

Table 2: Matched predictor comparison on Qwen3-Next-80B-A3B ($P = 107$, 16 held-out requests). R is candidate Recall@P (%) and C candidate turnover, new experts per decode-layer event. The same-turnover columns give FRAME recall within each reference's budget; the same-recall columns the minimum FRAME turnover reaching its recall.

	Reference		Matched turnover		Matched recall	
Predictor	Recall@P (%)	Candidate turnover	FRAME Recall@P	Recall gain	FRAME turnover	Turnover reduction
FRAME ranking	89.63	22.84	–	–	–	–
Smoothed route Markov	88.30	21.31	89.54	**+1.23**	13.84	**+35.0%**
Route-only Markov	88.26	21.27	89.54	**+1.28**	13.84	**+34.9%**
ProMoE	84.26	17.24	89.01	**+4.75**	6.24	**+63.8%**
MoE-Infinity EAM	80.97	22.80	89.63	**+8.66**	5.28	**+76.8%**

Against smoothed route Markov, FRAME gains 1.23 recall points at the same candidate-turnover budget (paired 95% interval $[0.76, 1.74]$; 15 of 16 requests improve) and needs 35.0% less turnover to reach its recall (Appendix C.1).

The predictor frontier does not by itself determine throughput: candidates may be rejected, arrive late or be evicted before use. We next trace the ranking through the live residency path.

5.2 EXPERT-MIGRATION TRAFFIC AND THROUGHPUT

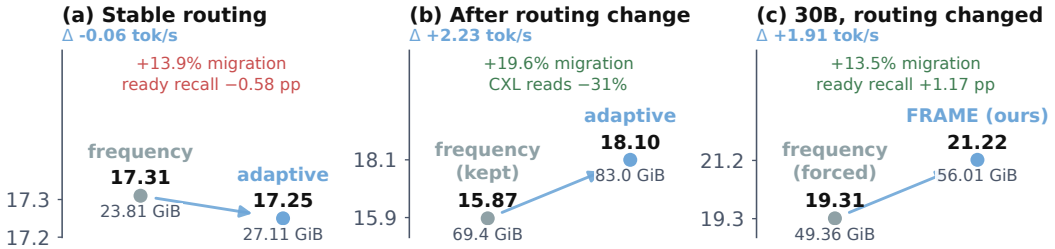

Figure 4: Effect of expert migration under different routing conditions. Each panel is one matched pass: its lower-migration arm on the left and its higher-migration arm on the right (grey and blue), the completed migration printed under each marker, the throughput change under the panel title, and one mechanism metric at the top. Each arm is named by the ranking that decides residency — frequency, the adaptive ranking, or FRAME's own selection — and the parenthetical says how that state arose: (b) keeps the frequency ranking after the shift, and (c) forces it. Extra migration has no observed payoff under stable routing, but coincides with fewer in-place CXL reads and higher throughput after the routing change, and with higher ready recall and higher throughput on the 30B geometry.

A ranking gain does not translate into throughput by itself. The runtime benefit depends on whether the experts a ranking promotes stay useful long enough to amortize the migration they cost. Under stable routing, aggressive adaptation raises turnover without reducing CXL demand, because the same experts would have been served in place; after a routing change, the same mechanism can cut repeated in-place reads by replacing future CXL traffic with reusable DDR5 residency.

When migration pays. Additional migration helps only when it converts misses into timely reuse: after the routing change the arm that moves more bytes cuts in-place CXL reads by 31%, and on the 30B it raises ready recall by 1.17 points (Fig. 4b–c). Migration volume alone does not order throughput.

Online feedback. The loop that sets the blend weights regulates how aggressively the adaptive ranking is acted on, rather than maximizing predictor accuracy: its value is the migration it avoids, not the throughput it adds. On one 80B build it removes 16.6% of the bytes the ranking controls at a 10 GiB pool (22.74 to 18.97 GiB/run) and 11.3% at 5 GiB, and roughly halves late copies, with ready recall unchanged or marginally higher; throughput moves the same way but only by 0.075 tokens/s at 10 GiB (paired interval $[+0.006, +0.144]$) and 0.125 at 5 GiB ($[+0.056, +0.194]$). The effect doubles when the pool is halved, so we read it as bandwidth discipline on the shared link (Appendix D.1).

Frequency-core sensitivity. A fifteen-point sweep of the engine core shows that F_{eng} mainly controls migration: ready recall stays inside 83.3–84.4% while completed migration moves by a factor of 2.8, and throughput is flat over $F_{\text{eng}} \in [54, 100]$ but 1.78 tokens/s (10.7%) lower at $F_{\text{eng}} = 0$, with $3.6\times$ the migration (Appendix D.3).

6 CONCLUSION

CXL makes a missing expert directly executable, turning expert migration from a prerequisite into a bandwidth investment. FRAME uses low-migration frequency residency under stable routing, switches to the adaptive ranking when the routing distribution shifts, and bounds the migration traffic that adaptation creates, reaching 17.09 tokens/s on the matched 80B stream, 17.6% above the strongest adapted baseline.

Decode throughput depends on whether a useful ranking converts into timely ready reuse while reducing expensive in-place reads. More generally, in a system with directly executable slow memory the question is no longer how to eliminate misses, but how to allocate the limited fast-memory bandwidth between residency and access: a migration is worth paying for only when the demand traffic it avoids outweighs the copy traffic and contention it introduces.

AI USE STATEMENT

AI assistance was used to draft and edit this manuscript and to organize the evidence package. All experimental results were produced by the authors' own code and runs on the hardware described in Section 4.1; no reported measurement was generated by a language model. The authors have verified the claims, citations, and provenance of the code and data reported here.

REPRODUCIBILITY STATEMENT

The measurements were produced by a modified llama.cpp CPU build (Section 3.4) driven by scripts that fix the model checkpoint, prompts, seed, thread and NUMA binding, and repetition count. The evaluated model bytes are identified in the supplement by their SHA-256 (`31cd9bce2e069d92...`); they were downloaded as a Q4_K_M GGUF of Qwen/Qwen3-Next-80B-A3B-Instruct and the upstream repository revision was not recorded at download time. The exact binding is `numactl --cpunodebind=1 --membind=1` with 24 threads on CPUs 28–51 and queue depth eight, with copy-path worker threads placed at CPU 51, which overlaps the inference range; Appendix A.1 reports a dedicated-core control that moves the worker off that range and leaves the comparison ordering unchanged. A build-to-experiment mapping and the online-feedback implementation are documented in Appendix D.1. The controlled regime stream uses a 32-token generation cap; the LongBench expansion uses a 120-token cap, under which the model's actual generation frequently ends near 32 tokens on its own. A trace harness reproduces the policy half of the runtime identically, so the trace-level comparisons of Section 5.1 exercise the same rule as the live engine. The estimator and the comparison rule are stated in Appendix B.1. Runner scripts, the reported campaign configurations, per-request aggregates, and the device counter collection used for the CXL traffic figures will be released as an anonymized supplement. LongBench (Bai et al., 2024) supplies the held-out expansion requests; truncation to 16,000 characters is applied by our harness and is described in Section 4.1. Each deviation from the published decision rules is listed in Appendix A.1.

ETHICS STATEMENT

This work concerns inference efficiency for existing language models and does not alter their capabilities or their misuse risks. It raises no human-subjects or privacy concerns; the workload consists of public benchmark passages. Released artifacts will exclude credentials, private prompts, network addresses, and author-identifying infrastructure details.

REFERENCES

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3119–3137. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.172. URL https://aclanthology.org/2024.acl-long.172/.

Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, Yuening Zhu, Qingliang Ou, Jiaqi Liao, Xianglin Chen, Zhiyuan Ai, Yongwei Wu, and Mingxing Zhang. KTransformers: Unleashing the full potential of CPU/GPU hybrid inference for MoE models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, 2025. doi: 10.1145/3731569.3764843. URL https://github.com/kvcache-ai/ktransformers.

Artyom Eliseev and Denis Mazur. Fast inference of mixture-of-experts language models with offloading, 2023. URL https://arxiv.org/abs/2312.17238v1.

Zhiyuan Fang, Zicong Hong, and Yuegui Huang. Fate: Fast edge inference of mixture-of-experts models via cross-layer gate, 2025. URL https://arxiv.org/abs/2502.12224v2.

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. URL https://arxiv.org/abs/2101.03961.

Xin He, Shunkang Zhang, Kaijie Tang, Shaohuai Shi, Yuxin Wang, Zihao Zeng, Zhenheng Tang, Xiaowen Chu, Haiyan Yin, Ivor W. Tsang, and Yew Soon Ong. ExpertFlow: Efficient mixture-of-experts inference via predictive expert caching and token scheduling. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference (DAC)*, 2026. doi: 10.1145/3770743.3804292. URL https://arxiv.org/abs/2410.17954v2.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *ACM/IEEE International Symposium on Computer Architecture*, 2024. URL https://www.microsoft.com/en-us/research/publication/pre-gated-moe-an-algorithm-system-co-design-for-fast-and-scalable-mixture-of-expert-inference/.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2402.07033v3.

Taehyun Kim, Kwanseok Choi, Youngmock Cho, Jaehoon Cho, Hyuk-Jae Lee, and Jaewoong Sim. MoNDE: Mixture of near-data experts for large-scale sparse models, 2024. URL https://arxiv.org/abs/2405.18832v1.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2006.16668.

Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pengfei Zuo, Tongping Liu, Sheng Yang, Juncheng Yang, K. V. Rashmi, et al. Pond: CXL-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2023. doi: 10.1145/3575693.3578835. URL https://arxiv.org/abs/2203.00241.

Qingxiu Liu, Cyril Y. He, Hanser Jiang, Zion Wang, Alan Zhao, and Patrick P. C. Lee. FluxMoE: Decoupling expert residency for high-performance MoE serving, 2026. URL https://arxiv.org/abs/2604.02715v2.

Qwen Team. Qwen3-Next-80B-A3B-Instruct: Model card, 2025. URL https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct. Accessed 2026-09-14; the upstream repository revision was not recorded at download time, and the evaluated GGUF is identified by SHA-256 in the artifact manifest.

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. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1701.06538.

Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. ProMoE: Fast MoE-based LLM serving using proactive caching, 2025. URL https://arxiv.org/abs/2410.22134v3.

Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. PowerInfer: Fast large language model serving with a consumer-grade GPU, 2023. URL https://arxiv.org/abs/2312.12456v2.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. HOBBIT: A mixed precision expert offloading system for fast MoE inference, 2024. URL https://arxiv.org/abs/2411.01433v2.

Wenxin Wang, Yule Hou, Yu Ji, Peng Qu, and Youhui Zhang. Achieving cloud-grade SLOs for local mixture-of-experts inference through CPU–GPU hybrid design. In *Proceedings of the 20th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2026. URL https://arxiv.org/abs/2606.10493.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Efficient MoE inference on personal machines with sparsity-aware expert cache, 2025. URL https://arxiv.org/abs/2401.14361v3.

Xinjun Yang, Qingda Hu, Junru Li, and Feifei Li. Beluga: A CXL-based memory architecture for scalable and efficient LLM KVCache management, 2025. URL https://arxiv.org/abs/2511.20172v2. SIGMOD 2026.

Yuchen Yang, Yaru Zhao, Pu Yang, Shaowei Wang, and Zhi-Hua Zhou. ZipMoE: Efficient on-device MoE serving via lossless compression and cache-affinity scheduling, 2026. URL https://arxiv.org/abs/2601.21198v1.

Dongha Yoon, Younghoon Min, Hoshik Kim, Sam H. Noh, and Jongryool Kim. TraCT: Disaggregated LLM serving with CXL shared memory KV cache at rack-scale, 2025. URL https://arxiv.org/abs/2512.18194v1.

Hanfei Yu, Xingqi Cui, and Hong Zhang. Taming latency-memory trade-off in MoE-based LLM serving via fine-grained expert offloading, 2025. URL https://arxiv.org/abs/2502.05370v2. EuroSys 2026.

Zeyu Zhu, Gang Li, Peisong Wang, Zitao Mo, Minnan Pei, Zhuoran Song, Xiaoyao Liang, and Jian Cheng. DALI: A workload-aware offloading framework for efficient MoE inference on local PCs. *arXiv preprint arXiv:2602.03495*, 2026. URL https://arxiv.org/abs/2602.03495.

A Experimental details and provenance

A.1 Baseline comparison scope

Table 3: Actual comparison scope of the reported baselines. Every row is a mechanism adaptation to the CPU–CXL setting unless noted, not a reproduction of the original system.

Source paper	What is compared	Important limitation
KTransformers (Chen et al., 2025)	Fixed profiled expert residency, by frequency, uniform and random placement	Profile quality and byte budget must match
Fiddler (Kamahori et al., 2025)	Hot set resident with cold experts computed in place; reuse-threshold copy admission	Not its complete latency model
Mixtral-Offloading (Eliseev & Mazur, 2023)	Demand admission and LRU replacement	Not the full original offloading runtime
ProMoE (Song et al., 2025)	Preemptive migration scheduling	Not its learned predictor plus full runtime
MoE-Infinity (Xue et al., 2025)	Activation-pattern ranking/cache adaptation	CUDA execution replaced by CPU–CXL
ExpertFlow (He et al., 2026)	Offline replay predictions consumed by the runtime	Online predictor cost not included
Pre-gated MoE (Hwang et al., 2024)	Earlier gate information in an unchanged model	Not a retrained Pre-gated MoE

Uniform execution conditions. Table 3 is a scope table, not a measurement table. Within each policy comparison pass the policy-bearing arms share the same binary, the same expert-byte accounting, the same copy primitive (`MCCOPY` with a fixed pool and group 4), the same queue depths (8/8), the same prefetch worker core, the same NUMA binding, the same warm-up and the same tuning budget; the runners share one `base_env` block and one invocation line verbatim, so only the listed mechanism differs between arms. The All arms use the same tuning budget without per-policy retuning. Two implementation details differ across the arms. The copy worker core (51) lies

inside the compute range (28–51), so arms that copy more also contend more with themselves. A dedicated-core control moves the worker to 52, the same NUMA node but outside the inference set, and leaves the comparison where it was: on the 14-request stream FRAME is 17.11 tokens/s post-shift against 16.19 for fixed frequency and 14.50 for Pre-Gated, gaps of $+0.92$ $[+0.56, +1.29]$ and $+2.61$ $[+2.24, +2.98]$ over two passes each, against $+0.90$ and $+2.64$ with the deployed placement, so the dedicated-core control preserves the ordering reported in the system tables. The second condition that is not uniform is the Pre-gated row, which reuses the released checkpoint rather than retraining, so its ranking information is weaker than in the original system.

The frequency policy in the pool-budget sweep is capacity-matched, which is not the configuration of the deployed 10 GiB pass, where the profiled top-107 experts per layer are the set that fits the pool. A budget sweep cannot hold that set fixed: it is 5136 experts, against pools of 2631, 4211, 7895 and 10527 experts at 5, 8, 15 and 20 GiB—$1.95\times$ and $1.22\times$ over the first two, and 35% and 51% under the last two. The four sweep points were therefore re-measured with the hot set cut to the per-layer capacity the same budget rule affords, $107B/10$ experts per layer for a pool of B_{pool} GiB—53, 85, 160 and 214, filling about 97% of every pool, and each a prefix of the one above it in a single frequency ranking, so the row is one policy at four capacities rather than four policies: at 20 GiB the matched set reaches 17.18 tokens/s where the fixed set was flat at 14.08, and FRAME remains above it at all four budgets. Scaling the pool from 5 to 20 GiB reduces the gap to all-DDR5 from 23% to 5.9%, and the frequency baseline scales its hot set with the same budget.

A.2 Expert-migration break-even microbenchmark

The break-even comes from a standalone microbenchmark (`breakeven_bench.c`) that allocates its own buffers and runs no system policy. Its definitions are as follows.

Unit and buffers. One unit is one expert's worth of bytes, $E = 2\,039\,808\,\mathrm{B} = 1.9453\,\mathrm{MiB}$ — the engine's own printed per-expert size (gate 576 KiB + up 576 KiB + down 840 KiB), taken from the run log rather than assumed. Each of the two nodes holds an anonymous 2 GiB buffer, $16E$ so that a pass over it cannot be served from cache; the binary refuses to run below $8E$. Pages are placed with `set_mempolicy(2)` before first touch, the same binding the engine's expert pool uses, with the CXL buffer bound to one node and the DDR5 buffer to the other, and the process is started under `numactl --cpunodebind=1 --membind=1`.

Timing. All three passes run at $T = 24$ threads, the thread count of the generation protocol, because one thread saturates neither link: at $T = 1$ the DDR5/CXL ratio reads 0.70 rather than 0.088 and N^* reads 2.56. The *in-place CXL read* is a traversal of every byte of the CXL buffer, one 8-byte word at a time; the *resident DDR5 read* is the same traversal of the DDR5 buffer; the *migration* is a `memcpy` of the same byte count from the CXL buffer into the DDR5 buffer — the copy path the engine's own migration uses — with one thread per 64-byte-aligned slice. Each value is the *median* of 7 full passes after one discarded warm-up pass, and the dispersion quoted is the max−min spread across those passes (0.19–1.36 μs per unit, 1.4%–3.4% of the median).

Break-even. Over N reuses of one unit, never migrating costs $N\,T_{\mathrm{cxl}}$ and migrating once costs $T_{\mathrm{prom}} + N\,T_{\mathrm{ddr5}}$, so the two cross at

$$N^* = \frac{T_{\mathrm{prom}}}{T_{\mathrm{cxl}} - T_{\mathrm{ddr5}}}, \tag{9}$$

and the measured terms are $T_{\mathrm{cxl}} = 137.52\,\mu$s (14.83 GB/s), $T_{\mathrm{ddr5}} = 12.08\,\mu$s (168.87 GB/s) and $T_{\mathrm{prom}} = 140.58\,\mu$s (14.51 GB/s, read+write), giving $N^* = 140.58/(137.52 - 12.08) = 1.12$. N^* is a continuous reuse count; in discrete execution it corresponds to at least two complete reuses of the migrated unit.

Two checks support reading this measurement as link-dominated. T_{prom} lands within 2.2% of T_{cxl}, because both are bounded by the same CXL link, so $N^* \approx 1/(1-T_{\mathrm{ddr5}}/T_{\mathrm{cxl}})$: the agreement is consistent with a link-dominated regime on our system rather than with a CPU-compute bottleneck. And the migration bandwidth reproduces the engine's own measured migration bandwidth ($\sim$14.7 GB/s) to 1.3%, so the microbenchmark's copy path is the one the deployed system pays. Panel (a) of the same figure is not from this benchmark: it reuses the all-CXL and all-DDR5 endpoints of the same 10 GiB pass.

Table 4: Expert-migration break-even microbenchmark, raw timings per expert unit ($E = 2\,039\,808$ B). Median of 7 full passes (5 at $T = 1$); spread is max−min across passes. The $T = 24$ rows are the ones the break-even uses; the $T = 1$ rows are the control that fixes the thread count.

Threads	Pass	μs per unit	GB/s	Spread (μs)
24	in-place CXL read	137.52	14.83	1.356
24	resident DDR5 read	12.08	168.87	0.191
24	migration (read+write)	140.58	14.51	0.461
1	in-place CXL read	302.34	6.75	2.675
1	resident DDR5 read	212.64	9.59	0.660
1	migration (read+write)	229.59	8.88	1.320

A.3 Residency freeze in the reported system rows

The reported FRAME configuration includes an evaluation-time residency freeze after 30 issue calls; it is not part of the deployed rule of Table 16, which ends with the engine's eviction hysteresis. One issue call is made per generated decode token, so runs on these prompts end after about 32 tokens against a 120-token budget and the freeze covers only the final one or two tokens; the adapted baselines keep mutating throughout. The freeze is ablated directly on the online-feedback build, where the paired throughput difference is $+0.050$ tokens/s with an interval that spans zero. Longer generations would be required to resolve it.

The volume columns report the bytes moved in one run, summed over the expert's three weight tensors (gate, up, down).

A.4 Regime-selector calibration

The threshold and the saturation are the two ends of the score band the selector was calibrated on, and the reported runs use the stable end, 0.770115. The calibration is a preceding pass over the same 14 request identities under the same protocol, run before the reported runs, so it is a calibration pass rather than a held-out threshold selection: the threshold is the nearest-rank 95th percentile of the six stable-routing scores, so the mode decision itself uses no post-switch outcome; the saturation is the nearest-rank 5th percentile of the eight post-switch scores, and it only sizes the graded suffix, which is a width diagnostic rather than a second state. On that protocol the stable-routing scores read 0.654–0.770 and the post-switch scores 0.834–0.862, so the two distributions leave a 0.064-wide band and no request of the reported stream falls inside it; the mode sequence is therefore identical for every threshold in the band. Re-measuring the stream with the threshold at the band's midpoint (0.801947, two passes of each of the three compared policies) leaves the ordering and the gaps where they were: FRAME leads fixed frequency by $+0.97$ $[+0.60, +1.34]$ and Pre-Gated by $+2.70$ $[+2.31, +3.08]$ on the eight post-transition requests, against $+0.90$ $[+0.53, +1.28]$ and $+2.64$ $[+2.26, +3.01]$ at the deployed threshold, so the choice within the band does not carry the result. The held-out campaign of Table 7 neither selects nor retunes either constant: it runs the same controller with the same frozen threshold and increment margin, and its runner overrides no selector parameter (only the capacity preset), so those 24 requests report what the frozen selector does on an unseen workload rather than a tuned one. That campaign does not enable the engine's per-request regime trace, so its selector behavior is reported through system-level outcomes only.

A task-shift control separates semantic task changes from routing displacement. It uses six GSM8K prompts the reported stream does not use, then four unused MMLU and four unused HumanEval prompts. The post-change requests score 0.628–0.687, inside the range the six stable-routing requests of the same control stream occupy (0.610–0.774), so d_r never crosses the deployed threshold and the controller keeps the frequency ranking from the third request on, including every post-change request. That is the intended behavior rather than a missed transition. The selector is not a task-change or domain classifier; it measures how far the incoming routes have moved from the long-term frequency ranking and makes the adaptive ranking eligible only when that displacement exceeds the calibration range. A positive measured prompt-hit advantage condition of Section 3.3 is still required before the mode changes. A semantic workload change need not induce an actionable residency change, and on this control stream it does not.

B ADDITIONAL SYSTEM RESULTS

B.1 STATISTICAL PROTOCOL

This appendix states the statistical protocol behind the reported points and the rule that limits comparisons to one pass.

For the four-request ablation and sensitivity passes, a point estimate is the median of three repetitions per request, averaged over the four request medians. The standard deviation is over the twelve runs and is a dispersion measure rather than a confidence interval on the mean of request medians; the three repetitions quantify within-request variability, so inferential comparisons use the four request-level pairs (t interval over $n = 4$). The long-context expansion and the 30B control use their own estimators, stated with their tables (Appendices B.4 and B.5).

Two passes are compared only within themselves: the deployed configuration is re-measured in every pass that needs it, and only within-pass comparisons are drawn on.

B.2 OUTPUT EQUIVALENCE WITH THE ENGINE-FREE REFERENCE

Greedy decoding (temperature 0, seed 42) on the four held-out prompts, three reference repetitions each, with the deployed configuration's diagnostics captured on a separate stream so they cannot interleave with the text. The rendered outputs match the engine-free reference on every run. This is a *text* comparison: it is consistent with token-identical decoding but does not establish it, since different token sequences can render the same text and this engine prints no token ids, and it does not establish task quality, for which the official LongBench scorer was not run.

Table 5: Output equivalence with the engine-free reference.

Prompt	Identical runs	Prompt / output chars
qasper_0	3/3	1450 / 1492
qasper_1	3/3	1339 / 1381
gov_report_0	3/3	1443 / 1485
gov_report_1	3/3	1365 / 1407

B.3 CONTINUOUS REGIME-STREAM BREAKDOWN

The per-task means underlying Fig. 3 are shown below.

A second segment-level pass measures residency and traffic with the regime enabled, one process per segment per policy, so the engine's end-of-run report belongs to that segment alone. On the six stable-routing requests FRAME reaches 85.06% ready recall at 23.79 GiB of copy volume per stream, against 85.10% and 23.81 GiB for the frequency ranking and 84.52% and 27.11 GiB for adaptive: forcing adaptive on stable routing costs 13.9% more traffic for half a point less recall. After the transition it lands between the two rankings, at 16.92 tokens/s and 33.58 GiB against 16.76/29.62 for frequency and 17.12/34.59 for adaptive.

Replicate passes. The four policies of Section 4.2 were each run three more times on the same 14 requests, same binary, pool and bindings, so the headline comparison carries two different intervals: one whose unit is a whole stream, and one whose unit is a request. Per-replicate post-shift means are 17.18/17.18/17.09 for FRAME, 16.24/16.23/16.27 for fixed frequency, 17.25/17.39/17.35 for fixed adaptive and 14.51/14.47/14.56 for Pre-Gated. *Replicate level* ($n = 3$ streams, two-sided 95% t with two degrees of freedom): FRAME is $+0.90$ tokens/s ahead of fixed frequency ($[+0.70, +1.10]$) and $+2.64$ ahead of Pre-Gated ($[+2.38, +2.89]$), and trails fixed adaptive by 0.18 ($[-0.43, +0.07]$, i.e. the two are not separated at this resolution). *Per-request level* (the eight post-transition requests, paired inside each pass and then averaged over the three passes, $n = 8$): $+0.90$ ($[+0.53, +1.28]$, faster on 8 of 8 requests), $+2.64$ ($[+2.26, +3.01]$, 8 of 8), and 0.18 ($[0.14, 0.22]$, faster on 0 of 8). The stream-level interval is the one that answers run-to-run uncertainty; the request-level interval shows the gap is not carried by a few requests. Both rest on the same three passes, and neither depends on the single pass of Table 6.

Table 6: The continuous 14-request 80B comparison, one pass per policy; the throughput column is decode throughput in tokens/s. GSM8K has six requests; MMLU and WMT16 have four each. Shift is the mean over the eight MMLU and WMT16 requests. Every row uses the same binary, prompts, model, 32-token limit, and 10 GiB pool. The four policies compared in Section 4.2 were then repeated three more times each; the per-replicate means and the paired per-request intervals in the text come from that campaign, and its passes agree with this one to 0.1 tokens/s. The deployed policy holds the frequency ranking on the six GSM8K requests and the adaptive ranking on the MMLU and WMT16 requests, so its row reads with the fixed-frequency control in the first column and with the fixed-adaptive control in the next two, which is what Fig. 3(a) plots as one row for the stationary segment.

Policy	GSM8K	MMLU	WMT16	Shift
FRAME auto	17.13	17.14	17.03	17.09
Fixed adaptive	17.25	17.36	17.26	17.31
Fixed frequency	17.31	16.45	15.91	16.18
Pre-Gated	14.50	14.50	14.55	14.52
ProMoE	14.97	14.52	14.11	14.31
Static-Freq	12.90	13.07	15.20	14.14
Uniform	12.86	12.74	12.81	12.78
Cost-Gated	12.81	12.74	12.84	12.79
LRU	12.90	12.74	12.70	12.72
EAM	15.14	11.44	10.76	11.10
All-CXL	11.14	11.14	11.15	11.14

B.4 Held-out workload expansion

The controlled regime stream of Section 4.2 is intentionally small and structured to expose a domain transition. We therefore run a broader held-out LongBench campaign on **24 requests drawn from six tasks** (`qasper`, `gov_report`, `narrativeqa`, `multifieldqa_en`, `hotpotqa`, `2wikimqa`), none sharing a hash with the controlled stream, at the published 16,000-character truncation with two repetitions per request. The campaign uses the same engine (SHA-256 `fb9a40d36e033606...`), NUMA binding, 10 GiB pool, seed, temperature and a 120-token generation cap.

Table 7: Held-out workload expansion, 24 requests over six LongBench tasks, paired per request. "Δ vs FRAME" is the per-request paired difference and its t interval over the 24 request pairs, so a negative value is a baseline behind FRAME. All policy-bearing rows and All-CXL are measured in one online-feedback-active campaign. An arm that does not maintain the engine's residency ledger has no ready-recall entry, marked "–" rather than reported as zero.

Configuration	tok/s	ready recall %	Δ vs FRAME	95% interval
All-DDR5 (reference)	19.28	–	–	–
FRAME	**16.92**	86.76	–	–
Pre-Gated	14.38	87.75	-2.540	$[-2.633, -2.446]$
Static-Freq	13.98	–	-2.935	$[-3.347, -2.523]$
All-CXL (direct reads)	10.67	–	-6.248	$[-6.466, -6.030]$

FRAME leads the strongest adapted baseline by $+2.54$ tokens/s ($+17.7\%$, interval $[+2.45, +2.63]$), and it leads on each of the six tasks ($+2.38$ to $+2.74$ tokens/s), with the online loop active. The requests differ in routing: the per-request oracle top-P recall (the recall of the request's hindsight top-P expert set) at the deployed capacity spans 76.4–88.0%, against the $\sim 82.8\%$ on which the four-request headline pass happens to sit. The static arm does not compute the oracle line, so the signature is read from the deployed arm and those runs are excluded rather than averaged in.

B.5 Matched regime control on 30B

Unlike the broader sensitivity analyses of this appendix, this campaign changes only the selected regime state, with all residency settings fixed. All arms use the same final binary, 4 GiB pool, $P = 29$, 16:13 dynamic-to-frequency split, prompts, copy path, NUMA binding and generation cap. The state is forced by one environment variable: with the override unset the binary reproduces the deployed configuration's outputs and routing state on the same prompts, verified request by

request. Each session is one continuous 12-request stream (4 GSM8K, 4 MMLU, 4 WMT16; 32 generated tokens each) in a fresh process, three sessions per arm in repetition-major order, and the unit of comparison is the request median over the three sessions.

Table 8: Matched regime control (Qwen3-30B-A3B, 4 GiB pool, $P = 29$, $D_{\text{eng}}{:}F_{\text{eng}} = 16{:}13$). "GSM8K prefix" is requests 1–4 and "MMLU–WMT16" requests 5–12; the names are descriptive, since this geometry has no stable prefix (see below). Ready recall, migration volume and late arrivals are per-session engine summaries over the twelve requests, medianed over the three sessions; the paired interval is against the deployed configuration on identical prompts, over the twelve request medians. Positive paired differences mean that the deployed FRAME is faster than the corresponding row. The static arm runs the engine's static-only mode, which issues no migrations and therefore reports no residency counters.

Policy	Tokens/s	GSM8K prefix	MMLU–WMT16	Ready recall %	Migration (GiB)	Late/tok.	Paired CI
FRAME (auto)	21.22	21.08	21.29	81.01	56.01	0.54	—
FRAME (forced freq.)	19.31	20.36	18.79	79.84	49.36	0.41	$[+1.13, +2.68]$
FRAME (forced adapt.)	21.23	21.07	21.31	81.01	56.00	0.54	$[-0.02, -0.00]$
Static-Freq	21.19	27.62	17.98	—	—	—	$[-3.40, +3.45]$

Forcing the frequency state costs 1.91 tokens/s against the deployed configuration (paired $[+1.13, +2.68]$ over twelve requests, in every one of them), and the cost is larger after the transition ($+2.50$ on requests 5–12) than before it ($+0.72$): on this geometry the adaptive state repays its 6.6 GiB of extra migration. Forcing the adaptive state reproduces the deployed row to 0.01 tokens/s, so the sign belongs to the state and not to the switch.

Against the adapted static baseline the overall mean is a tie ($+0.03$, $[-3.40, +3.45]$), and the tie is made of two different regimes. The static arm is 6.55 tokens/s faster on requests 1–4 (paired $[-13.01, -0.08]$, all four), because its hot set was profiled on GSM8K decode traces—the task of that prefix—and it pays nothing at all for residency, and 3.31 slower on MMLU–WMT16 (paired $[+2.23, +4.39]$, all eight) because nothing re-ranks it. It is a favorably profiled reference rather than a task-blind one. Which of the two carries the comparison depends on whether this geometry has a stable regime at all, which the next paragraph answers.

The deployed selector never enters the frequency state on this geometry. The quantity it acts on is the prompt-to-frequency distance of Eq. (6), medianed over layers: it is 0.79–0.86 (median 0.83) on every one of the twelve 30B requests, the first four included, against 0.654–0.770 on the stable prefix of the 80B stream and 0.834–0.862 after its transition, with the deployed threshold at 0.770115. The nominally stable segment of this geometry is therefore as far from a frequency ranking as the 80B is after its transition, which is why the frequency state is never selected: the geometry has no stable regime, and the favorably profiled static baseline leads exactly on the prefix it was profiled for. The MMLU–WMT16 segment therefore provides the relevant out-of-profile comparison, where the adaptive state leads by 3.31 tokens/s.

C Predictor analysis

C.1 Cross-geometry predictor frontier

Table 9 reports the cross-geometry matched-condition comparison of the current multi-signal ranking with frequency-only and route-state controls; the per-trace frontier points are provided in the artifact. Both matched columns are cross-trace aggregates, so neither defines a paired interval; the final column reports FRAME's recall at the reference turnover (the reciprocal matched condition) and is descriptive rather than part of the two primary matched comparisons. The main-text published-predictor comparison is in Table 2.

On the three deployed geometries of that table, removing the predictor-side frequency anchor moves the replay points toward both higher recall and higher candidate turnover; the anchor's system contribution is measured by the engine-frequency sweep of Section 5.2.

Table 9: The predictor comparison, read at *matched condition* on 15 held-out traces spanning 7 geometries: every entry is anchored at FRAME's own operating point (R^*, C^*) and each reference arm is read off its own frontier at the same condition. "—" denotes a condition a frontier does not span, not a tie.

			Matched turnover C ref. recall (%)		**Matched recall R**	**FRAME R at ref. $C^\dagger$**
Geometry	P	**FRAME R^*/C^***	**freq.**	**route**	**route C reduction**	**our R at $C^\dagger$**
JetMoE-8B	6	81.3 / 0.55	77.4 (+3.9)	—	58.5%	—
Granite-3.1-1B-A400M	16	82.6 / 1.02	78.4 (+4.2)	20.7 (+61.9)	70.6%	51.5 (−26.9)
OLMoE-1B-7B	19	67.5 / 2.86	52.2 (+15.3)	32.7 (+34.8)	47.7%	35.9 (−16.4)
PowerMoE-3B	20	80.2 / 1.29	73.4 (+6.7)	19.5 (+60.7)	63.3%	57.4 (−16.0)
Qwen3-30B-A3B	29	76.6 / 2.66	68.7 (+7.9)	41.6 (+35.0)	52.6%	41.0 (−27.7)
Qwen1.5-MoE-A2.7B	30	73.0 / 4.54	57.1 (+15.9)	31.0 (+41.9)	58.7%	—
Qwen3-Next-80B-A3B	107	84.4 / 9.65	74.9 (+9.5)	54.3 (+30.0)	55.4%	41.5 (−33.3)

C.2 Engine-rule replay analysis

The frontier of Appendix C.1 is scored on candidate turnover C, which counts ranking replacements and is not a byte quantity. The cross-model table and the ablations of Section 5.2 are scored on bytes, so the predictor arms have to be re-scored the way the engine consumes them before the two readings can be put side by side. This section does that.

Every trace is replayed through the same frozen engine-equivalent policy, but each prediction is scored as the engine uses it: only the first D entries of the ranking are issued, they are unioned with the engine's causal frequency floor of size $P-D$ computed at that event, the union is the resident set, and recall is the one-token-lookahead recall of that resident set. This is the second of the three recall stages of Table 14: copies are credited when they are admitted, without a completion deadline, so the column is *replay-resident* recall and not the engine-measured ready recall that the system tables report. Logical churn M charges each newly admitted expert its stored byte size, so it is comparable across models in bytes. The split is the deployed per-model split (§4.1), and capacities are the benchmark capacities, so the Qwen3-30B trace runs at $P = 27$ here and at $P = 29$ in Appendix C.1.

Two things follow. First, C and M do not agree: on the 80B the frequency arm's C is three times lower than FRAME's while its M is identical, which is why the C frontier and the M frontier point in different directions on one and the same trace. Second, on this metric FRAME has the lowest M of the four arms on twelve of the thirteen traces and ties on the thirteenth, at 0.39–2.19 points of replay-resident recall below the per-model-calibrated arm, which pays 11.4–55.8% more M for those points. FRAME's predictor weights use no model-specific offline calibration in this replay, so what is compared here is run-time self-calibration against per-model offline fitting; the engine frequency split remains the deployed per-geometry preset. These are aggregates over all scored events, so no paired interval is defined.

D Ablations and sensitivity

D.1 Online-feedback ablation

We isolate the online-feedback loop on build SHA-256 `fb9a40d36e033606...`, comparing the deployed controller with feedback disabled and with the static-frequency operating point at two pool sizes (Table 11). The online loop removes about a quarter of the bytes the ranking controls—the gate and up tensors—at both sizes, and roughly halves late copies, with ready recall unchanged or marginally higher. Total bytes moved per run fall by less, 16.6% at 10 GiB and 11.3% at 5 GiB

Table 10: Aggregate of the thirteen replayed traces by model geometry, under the engine's consumption rule: replay-resident recall % / logical churn M in GB per 1,000 decode events. "M lowest" counts the traces in which FRAME has the lowest M of the four arms (a tie with the frequency arm counts); the next column is the M saved against the per-model-calibrated arm and the last the replay-resident recall paid for it. P is the offline matrix's benchmark capacity, which is 21% of the expert count and not the deployed capacity of the system tables. The per-trace rows are in the artifact.

Model	P	Traces	M lowest	M vs. calibrated %	Recall paid (pp)
jetmoe	6	1	1/1	-22.7	0.92
granite31	16	3	3/3	-13.7 to -11.4	0.39–0.45
olmoe	32	2	2/2	-27.4 to -27.0	1.71–2.19
powermoe	20	2	2/2	-16.4 to -16.3	0.65–0.72
qwen1.5	30	2	2/2	-55.8 to -55.1	1.18–1.20
q3m30b	27	1	1/1	-39.4	2.17
q3n	107	2	2/2	-46.6 to -28.2	0.52–0.56

(22.74 to 18.97 and 21.67 to 19.23 GiB/run), because down-projection copies do not respond to the policy: their per-thousand-step volume is identical in the arms of this table, so they dilute a ratio without changing what the loop controls. Throughput moves with the same sign at both sizes and the effect doubles when the pool is halved: the online arm is 0.075 tokens/s faster at 10 GiB (paired interval $[+0.006, +0.144]$) and 0.125 faster at 5 GiB ($[+0.056, +0.194]$). Both intervals exclude zero, but the throughput effect is small relative to the reduction in migration; we therefore interpret the loop primarily as a migration-reduction mechanism, whose throughput value becomes visible once the pool is tight enough for the admission bound to bind. At 5 GiB the online-feedback policy lands on the static-frequency operating point—14.83 against 14.83, interval $[-0.11, +0.11]$, at equal recall and equal migration volume; at 10 GiB the frequency arm is 0.05 tokens/s ahead ($[-0.19, +0.09]$ against the online arm). The online loop therefore reduces migration without giving up the frequency baseline, and it is the migration traffic that separates the two policies at 10 GiB, not the throughput.

Table 11: Online-feedback ablation at two pool sizes (80B, 4 prompts $\times$ 3 repetitions per arm; paired per-prompt interval, where a negative value means the row is slower than the online arm). The online and online-disabled rows share one session; the daggered frequency-only rows come from a second session on the same build and the same four prompts, whose own online control measured 16.60 tokens/s at 10 GiB and 14.83 at 5 GiB. The two sessions' controls differ by 0.025 tokens/s (0.15–0.17%), which is this machine's session-to-session floor, and each daggered interval is computed inside its own session. Both sessions use the same instrumented online build, and every reported difference is computed within a session.

Pool	Configuration	tokens/s	Ready recall %	Migration volume (GiB/run)	Late/tok.	Paired CI
10 GiB	FRAME (online active)	**16.58**	85.47	18.97	0.43	—
10 GiB	FRAME (online disabled)	16.50	85.42	22.74	0.81	$[-0.14, -0.01]$
10 GiB	FRAME (frequency only)†	16.65	85.53	18.60	0.29	$[-0.09, +0.19]$
5 GiB	FRAME (online active)	**14.80**	76.58	19.23	0.49	—
5 GiB	FRAME (online disabled)	14.68	76.03	21.67	0.73	$[-0.19, -0.06]$
5 GiB	FRAME (frequency only)†	14.83	76.61	19.11	0.44	$[-0.11, +0.11]$

The sweeps of this and the following subsection isolate the engine-side frequency reservation under fixed policy constants, and we use them only to characterize the shape of the reservation–migration trade-off; Table 11 separately bounds the online-feedback effect in the tested configurations.

D.2 SENSITIVITY TO THE DEPLOYED FREQUENCY SPLIT

The logical-churn cost model checks the deployed preset rather than producing it: for each split F it fits the relation of Eq. (7), with $M(F)$ the logical churn, a fixed per-token time and b the fitted marginal logical-churn cost. Its two coefficients are fitted on this calibration pass and then frozen, at $a = 0.0552\,\mathrm{s}$ per token and $b = +2.12 \times 10^{-4}\,\mathrm{s}$ per unit of M, with b positive across all nine leave-one-out refits. The sweep below is the calibration pass; the validation pass re-measures it on prompts the coefficients were not fitted on and reproduces the shape at 16.55–16.675 tokens/s against 16.65–16.80. Applied blind to that pass, the frozen coefficients predict all fifteen points to a mean absolute error of 0.093 tokens/s against a run sd of 0.562, and Table 12 gives logical churn the lowest validation error of the five candidate inputs (0.093 tokens/s against 0.236–0.435). On the plateau itself the candidate inputs are not separated, and the optimum of the predicted curve, $F_{\mathrm{eng}} = 94$, is not separable from the rest of $F_{\mathrm{eng}} \in [77, 100]$, so the rule resolves an interval rather than a value: the deployed preset and the measured plateau both lie inside it.

Table 12: Five candidate inputs for the frequency split, each fitted on the same calibration rows ($1/\mathrm{tok/s} = a + b\,x$), frozen, and scored on the same independent validation rows. "Regret" is the measured throughput at the model's picked F_{eng} against the measured best point of that pass; it does not discriminate, because every non-constant input lands inside the plateau. "Off plateau" restricts the error to the eight validation points below $F_{\mathrm{eng}} = 54$, where the curve is still rising. F_{eng} and $D_{\mathrm{eng}} = P - F_{\mathrm{eng}}$ give identical errors because they are affinely related at a fixed capacity P.

Input x	**Mean |err|**	**Max |err|**	**Picks** F	**Regret**	**Off plateau**
constant (calibration mean)	0.435	1.442	0	+1.825	0.527
engine frequency core F	0.236	0.532	100	+0.125	0.242
dynamic slots $D_{\mathrm{eng}} = P - F_{\mathrm{eng}}$	0.236	0.532	100	+0.125	0.242
ready recall %	0.428	1.425	54	+0.125	0.516
logical churn M (GB)	**0.093**	**0.249**	94	+0.025	**0.072**

That pass's denser grid also resolves the plateau's upper end, where the table has only $F_{\mathrm{eng}} = 80, 94, 100$: the maximum is at $F_{\mathrm{eng}} = 87$. In the table, on the 80B the throughput rises with F_{eng} up to 54 and forms a broad plateau over $F_{\mathrm{eng}} \in [54, 100]$: the calibration sweep spans 16.65–16.80 tokens/s there and the disjoint validation sweep of Table 13 reproduces the shape at 16.55–16.675, statistically indistinguishable from the deployed $F_{\mathrm{eng}} = 77$ at $F_{\mathrm{eng}} = 80, 94, 100$; on OLMoE every additional frequency slot is neutral or worse at this pool size; on the 30B, 13 is the best measured point and 21 (the previous incumbent) is significantly worse.

D.3 ENGINE FREQUENCY-CORE SENSITIVITY

The deployed engine hands the policy $D_{\mathrm{eng}} = 30$ slots and reserves $F_{\mathrm{eng}} = 77$ for the top of its own online frequency estimate. We swept that split over fifteen points at the same operating point ($D_{\mathrm{eng}} + F_{\mathrm{eng}} = 107$, four held-out prompts $\times$ three repetitions, everything else fixed) to measure the effect of the reservation. Table 13 gives the answer: the reservation suppresses migration rather than increasing recall. Ready recall spans 83.34–84.35% — 1.01 percentage points, non-monotone, with the deployed 30:77 point 0.59 pp below the best point (D=13, same throughput, 5.1% fewer bytes) — while the moved bytes span 20.0 to 56.4 GiB per run, a factor of 2.8; giving the adaptive policy more slots incurs more migration while ready recall stays nearly flat. The volume column sums the gate, up and down tensors of every migration in one run.

Miss composition on the three-arm pass. On the same 14 requests the three arms agree on total miss to within 1.3% and `never` is 77–78% of it in every arm, so the never-resident miss component is nearly unchanged and the ranking reclassifies misses rather than removing them; these are single runs, so the readings are point estimates.

The pointwise prompt-level intervals resolve the high-F_{eng} end and $F_{\mathrm{eng}} = 64$, and cross zero at $F_{\mathrm{eng}} \leq 54$; four prompts with three repetitions each are four pairs and not twelve, and no multiplicity correction is applied to the fifteen comparisons, so we use these intervals only to bound the ready-recall variation at about one point rather than to order the middle of the plateau.

The miss taxonomy explains why. Of the experts each token needs, the share that is never resident and the share that was resident but had been evicted trade against each other as the split moves —

never $6.9 \rightarrow 11.7\%$ of selections, evicted $7.4 \rightarrow 3.8\%$ — while their sum stays at about 15.5%, and late arrivals stay below 2%. Changing which 107 experts are named therefore converts capacity misses into conflict misses and back; it cannot cover more of the router's mass, because the pool already holds the largest-107 part of it. A selection rule can only re-rank the tail, and the tail of that distribution carries roughly one point.

Implementation details. The frequency core is a preset, not a pinned set: the top-F experts are recomputed from the online estimate at every issue call and appended after the policy's D, so the two parts compete for the same slots and the reservation is a ranking priority rather than a guarantee. And the volume column above is per run; the per-thousand-step columns elsewhere divide it by the requested token budget (Appendix A.3), which is why this table reports the unambiguous unit.

Table 13: The deployed preset swept over fifteen points at the 80B operating point (4 held-out prompts $\times$ 3 repetitions each, 180 runs; paired against the deployed $D_{\text{eng}}{:}F_{\text{eng}} = 30{:}77$ on identical prompts, two-sided 95% t intervals over the four prompt-level pairs, each prompt being the mean of its three repetitions — the repetitions are not independent tasks). F is the engine's reserved frequency core and D the slots the policy ranks, $D + F = 107$. The miss columns are percentages of selected experts and sum with ready recall to 100%: "never" was never resident, "evicted" was resident and had been replaced before use, "late" was still in flight. "GiB/run" is the completed copy volume of one run, summed over the expert's gate, up and down tensors. In the last row the policy's cap is set to zero, which the engine reads as "no cap", so the policy ranks all 107 slots and no frequency core is reserved.

F	D	tokens/s	Ready recall %	Δrecall	never %	evicted %	late %	GiB/run
100	7	16.55	84.34	$+0.58$ $[+0.28, +0.88]$	11.6	3.8	0.11	20.33
94	13	16.64	84.35	$+0.59$ $[+0.36, +0.83]$	11.7	3.8	0.11	20.01
87	20	16.67	84.21	$+0.46$ $[+0.21, +0.70]$	11.6	4.0	0.11	20.10
80	27	16.65	83.92	$+0.16$ $[+0.12, +0.21]$	11.5	4.4	0.14	20.72
77	30	16.64	**83.75**	$+0.00$ $[+0.00, +0.00]$	11.4	4.6	0.15	21.08
64	43	16.62	83.35	-0.41 $[-0.69, -0.12]$	10.9	5.4	0.22	22.71
54	53	16.55	83.34	-0.42 $[-0.94, +0.11]$	10.3	5.9	0.29	24.53
49	58	16.43	83.38	-0.37 $[-0.98, +0.24]$	10.0	6.2	0.34	25.85
45	62	16.36	83.51	-0.25 $[-0.84, +0.34]$	9.7	6.3	0.38	27.18
40	67	16.20	83.68	-0.07 $[-0.56, +0.42]$	9.3	6.5	0.46	29.21
35	72	16.06	83.89	$+0.13$ $[-0.37, +0.63]$	8.8	6.6	0.56	31.57
28	79	15.87	83.91	$+0.16$ $[-0.57, +0.88]$	8.3	7.0	0.76	35.32
20	87	15.52	84.12	$+0.37$ $[-0.22, +0.96]$	7.6	7.2	1.00	40.94
10	97	15.18	84.23	$+0.48$ $[-0.08, +1.04]$	6.9	7.4	1.31	48.39
0	107	14.86	84.20	$+0.45$ $[-0.16, +1.06]$	6.3	7.7	1.72	56.44

D.4 Migration after a routing change

A prompt-only candidate construction repins frequency after the domain transition, which is the wrong state for the shifted traffic: on the final seven WMT16 requests it generates 15.87 tokens/s with 42.0k device read CAS events per sampled token, against 18.04 and 28.2k for FRAME and 18.10 and 29.0k for fixed adaptive. It moves fewer bytes over the whole run (69.39 against 82.96 GiB for fixed adaptive) yet reads more from CXL during this segment, which is why migration volume alone does not order throughput. The prompt-only and fixed-adaptive rows are one three-arm campaign on the build used by the 14-request stream (SHA-256 `316547e0...`); the FRAME row re-measures the same 22 requests (6 GSM8K, 4 MMLU, 12 WMT16) on the final binary (SHA-256 `047ce52a...`), whose prompt and output hashes match request by request, so that row is a one-pass descriptive comparison rather than a third arm of the same pass. The engine prints this counter as `GB` but divides by 1024^3, so the engine-reported migration volumes in this paper are given in GiB. The replay-side logical churn M of Table 14 is a separate, offline quantity and stays in decimal GB. The device-wide CAS counters sample the last 31 of each request's 32 generated tokens.

E METRICS AND EXECUTABLE RULE

E.1 DEFINITIONS OF THE REPORTED QUANTITIES AND SIGNALS

The main text defines the quantities of Section 3.1 and the signals of Section 3.2 in prose; the two tables below restate them in one place for reference.

Quantity	Stage	What it is
candidate recall	proposal issued	fraction of routed experts covered by the issued candidate set S, scored before any engine rule applies
replay-resident recall	admission applied	the same set after admission, with every admitted copy credited at once; it ignores completion deadlines, so it is an optimistic residency reading rather than a substitute for ready recall, and the optimism fails if a replay evicts an expert the runtime still serves before the replacing copy completes
ready recall	compute time	the set that is resident and valid when the layer actually runs; the only one of the three that says whether a CXL read was avoided
C	consecutive predictions	candidate turnover $\lvert S_t \setminus S_{t-1}\rvert$; no engine rule and no bytes
M	engine-rule replay	logical churn: each newly admitted expert charged its stored byte size, over the set the runtime actually builds
V	engine issue	submitted volume issued for migration in one run, summed over the expert's gate, up and down tensors
B	copy completion	completed migration bytes over the same three tensors; PMU-observed device traffic is larger, because it counts copies that do not finish

Table 14: Three recall stages and four update and traffic quantities. C is predictor candidate turnover; M is replay logical churn; V is submitted migration volume; B is completed migration bytes. These quantities are not interchangeable. Predictor frontiers use candidate recall and C; engine-rule replay uses replay-resident recall and M; live-engine ablations use ready recall together with V or B.

Signal	What it scores
lifetime frequency	activation count per expert over the run
window frequency	the same count restricted to a sliding window of recent routes
route-state	a posterior over the observed route-to-route counts, keyed on the previous token's route, plus a second-order term keyed on the previous two
first-order transition	the same table without that conditioning
gate logits (optional)	the previous token's gate output, used only for layers where it is available at the prediction point and matches the expert count

Table 15: The five ranking signals. Frequency is scored like the rest, but its share sets the width of the ranking anchor instead of entering the blend (Section 3.4).

E.2 THE DEPLOYED RULE, EXECUTABLY

Candidate-reset freshness threshold.

This is a second, independent statistic: the adaptive candidate is reset when the current request's profile differs from the cached candidate profile by more than the freshness threshold below, which is not the regime-selection distance of Eq. (6). The candidate-reset freshness test compares the candidate profile with the current request's prompt profile as a per-layer cosine distance and takes the median over layers. That distance is not scale-free: between two requests of the *same* task its median is 0.391 at 512 experts, 0.210 at 128 and 0.190 at 64, so the fixed threshold of 0.35 fires on 67.5% of same-task pairs in the first geometry and on 1.4% and 0.6% in the other two. On the 80B cross-task stream the same statistic still separates cleanly, 0.329 within a domain against 0.843 across one, so the quantity measures what it is meant to; it is its absolute scale that moves with the geometry. Measured on the engine rather than over traces, the deployed domain-drift stream sits at 0.51–0.53 throughout, above the threshold on every request, which is why the candidate is reset on every request there and the return to frequency is never reached.

Table 15 names the signals; Table 16 specifies their normalization and combination, the online weight update, the admission rule and the engine-side hysteresis needed to reproduce the deployed policy. Three properties are worth reading off it rather than inferring from the prose. The signals are combined by a *reward-weighted* blend whose weights are the pooled measured hit rates plus a flat 0.05 floor on each dynamic signal, so a signal that never wins still contributes a small share and the anchor cannot degenerate. Only the frequency signal's share sets the *anchor*; the other four are renormalized among themselves, which is why the anchor can sit strictly between zero and the full core. The admission budget is itself measured: the cap is $P(1-\bar{h})$ from the hit-rate EMA, so a set that is landing well admits less and lets the engine's retention hysteresis hold the pool stable. The retention bonus entering the policy's own ranking key is 0 in both the calibrated and the generic paths (window 16, bonus 0), so the 'retention' the ablation disables with `EVICT=0` is the engine's eviction hysteresis, not a policy bonus.

Table 16: Executable online ranking and admission rule within the selected regime; one decode prediction per layer. Every constant below is the value used in the reported runs. The shift threshold and the saturation are the two ends of the calibration band of Appendix A.4; the remaining selector guards (minimum prompt events, minimum valid layers, increment margin), the window, the EMA rate and the sample period are fixed settings shared across all three geometries, while the capacity P and the engine-side frequency split F_{eng} are per-geometry deployment inputs, the latter characterized in Appendix D.2 and swept in Appendix D.3.

Given E experts, capacity P, window $W = 16$, EMA rate $\gamma = 0.9$, sample period $K = 4$, minimum samples 16, admission seed $P/2$; **routing-aware selector:** minimum prompt events 1, minimum valid layers 24, shift threshold 0.770115, increment margin 0 (the saturation that brackets the calibration band is defined in Appendix A.4). **State per layer:** five count arrays over E experts, a deque of the last W routes, a transition table keyed by the previous route, and the scalars below.

1 $s_0 \dots s_4 \leftarrow$ lifetime frequency, route-state posterior, first-order transition, window frequency, gate logits (where the layer exposes them)
2 $q_s \leftarrow \textsc{Percentile}(s_s)$ over the E experts: rank fraction, ties share the lower rank, $\min = 0$, $\max = 1$
3 **if** online **and** at least one reward initialized: $w_s \leftarrow \bar{\rho}_s + 0.05$ for $s \in \{1,2,3,4\}$, renormalized to sum to 1; $w_0 \leftarrow \bar{\rho}_0 + 0.05$ as measured **else** $w_1..w_4 \leftarrow 0.25$ each (the frozen fallback)
4 $A \leftarrow \sum_{s=1}^{4} w_s q_s$ (the four dynamic signals; $w_0 q_0$ enters only through α below, never the blend); $d \leftarrow$ partial sort of A descending, ties by lower expert id, truncated to P
5 $\alpha \leftarrow w_0/(w_0 + \sum_{s\neq 0} w_s)$; $c \leftarrow \min(\text{round}(\alpha P), \lfloor 4P/5 \rfloor)$
6 $f \leftarrow$ the c highest lifetime-frequency experts (ties by lower id)
7 $S \leftarrow f$, then the experts of $d \notin f$ in order until $|S| = P$; if still short, lifetime-frequency order
8 $\kappa \leftarrow \text{clamp}(\text{round}(P(1-\bar{h})), 1, P)$ from the hit EMA (seed $P/2$)
9 $R \leftarrow \textsc{Admit}(S, R_{\text{prev}}, \kappa)$; issue migrations for $R \setminus \{\text{resident}\}$ in bulk-address order
10 **if** $t \bmod K = 0$: store $T_s \leftarrow$ the top-P experts of each signal scored *alone*, for the update below
11 **on** the realized route r_t: for every stored T_s, $\text{hit}_s \leftarrow |T_s \cap r_t|/|r_t|$; a signal that could not be scored is skipped, never credited
12 $\rho_s \leftarrow \gamma\rho_s + (1-\gamma)\,\text{hit}_s$ (first observation initializes); $\bar{\rho}_s \leftarrow$ exact running mean pooled over all layers
13 $D \leftarrow$ the prefix of the admitted ranking that the engine actually issued; $\bar{h} \leftarrow \gamma\bar{h} + (1-\gamma)\,|D \cap r_t|/|r_t|$ over that prefix
14 **engine, outside the policy:** with `EVICT=2`, eviction is attempted only when the pool exceeds its budget by 10%

The optional policy-side retention bonus is disabled ($b = 0$); retention in the reported system is provided only by the engine-side eviction hysteresis above.

F LIMITATIONS

Scope of the evidence. System measurements come from one CXL server (Section 4.1). The controlled regime stream is 6 stable-routing requests followed by 8 post-transition requests, the held-out expansion is 24 requests over six LongBench tasks, and the matched 30B state control is a twelve-request GSM8K–MMLU–WMT16 stream with three repetitions per arm; predictor replay spans further held-out traces and geometries. These measurements do not establish behavior across CXL generations or device models, CPU architectures, concurrent serving, or long generated sequences, and the DDR5 pool is not a total-host-memory limit. The output-equivalence reference is a quantized model, so full-precision quality preservation is not established, and full request latency, time

to first token and tail inter-token latency are not reported: the prompt phase is measurably slower under offloading, so the throughput ratios describe the decode phase only.

Deployment tuning. The deployed system still carries per-model dynamic/frequency settings (Section 4.1), so its online feedback has not removed all tuning. Those settings are presets rather than outputs of the rule: the logical-churn cost model that checks the engine core is calibrated on one prompt set per geometry, resolves a region of splits rather than a value, and on the 80B validation sweep the whole region lies inside the measured throughput plateau. The coefficients are not shared across geometries—the sign of the marginal logical-churn cost differs between the three models measured here—so whether an equally effective split can be selected without per-geometry calibration remains open.

Baseline and metric scope. The comparison set consists of mechanism adaptations rather than reproductions (Appendix A.1), and missing learned-predictor overhead, weak profiling, or unequal copying primitives can affect a baseline's strength in either direction, so no system table should be read as a reproduction of, or a claim over, a published system. Device-wide PMU counters cannot isolate a single job when other activity shares the device, so the CXL traffic figures are within-run comparisons rather than absolute bandwidth. Frontier points are selected on validation requests and reported on held-out ones, and C and M capture the decision the runtime made rather than the bandwidth the device delivered.