CACHE-AWARE JOINT ROUTER ADAPTATION FOR MEMORY-EFFICIENT MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) models activate few experts per token, yet their full expert sets can exceed GPU memory and require repeated weight transfers during decoding. We formulate expert-cache management as a model-side algorithmic problem and propose cache-aware post-training that jointly adapts the MoE backbone and lightweight auxiliary routers while preserving the native inference-time Top-K rule. The update-only **Temporal Router** learns same-layer retention across tokens without proactive loading. The full **Spatio-Temporal Router** adds a **Spatio Router** that uses the causal predecessor's hidden state to refine the temporal cache before target-layer access. We evaluate both modes on Qwen3 and GPT-OSS across GSM8K, MATH, and CommonsenseQA. Temporal Router consistently improves hit rate and reduces expert-weight traffic over matched LM-only baselines. On Qwen3, the full mode improves adjusted hit rate by 1.15–18.03 points and reduces traffic by 4.6–53.3% relative to the strongest evaluated prefetching baseline; GPT-OSS results are competitive but task-dependent. Auxiliary-only training preserves baseline accuracy but yields modest coverage gains; joint post-training achieves substantially higher coverage. Sensitivity analyses distinguish the effects of cache capacity, refinement budget, and cache-loss weight on coverage, traffic, and quality.[1].

1 INTRODUCTION

Mixture-of-Experts (MoE) models increase capacity by activating only a small expert subset per token. This computational sparsity does not eliminate weight-storage costs: when experts exceed an inference worker's GPU memory, decoding requires repeated transfers from slower memory. Cache management must decide which experts to retain and when to fetch them.

Existing approaches exploit expert-access locality through caching and prefetching. Classical LRU, LFU, and LRFU use recency or frequency, while MoE-specific methods exploit access traces, hidden-state predictors, retrieved expert maps, or persistent expert sets. Recent STEP and ST-MoE systems combine temporal and cross-layer signals for prefetching (Liu et al., 2026; Zhao et al., 2026). This motivates a different question: can post-training jointly adapt expert demand and cache priorities, rather than only predict or schedule an existing access pattern?

We address this question with a cache-aware objective that jointly adapts the MoE backbone and lightweight auxiliary routers. The **Temporal Router** scores resident experts for retention after each layer access, anticipating reuse at the same layer by later tokens. The optional **Spatio Router** uses the causal predecessor's hidden state to refine the cache before target-layer access. Native routing still selects the executed experts; auxiliary routing controls residency. The update-only Temporal Router incurs no proactive traffic, while the full **Spatio-Temporal Router** stacks bounded pre-access refinement on the same retention stage.

Across two MoE backbones and three reasoning tasks, Temporal Router improves hit rate and traffic over replacement policies on matched LM-only post-trained backbones. The full mode leads evaluated prefetchers on all Qwen3 tasks in adjusted hit rate and traffic, with task-dependent GPT-OSS results. Ablations support joint adaptation and temporal retention; budget sweeps reveal the coverage–traffic trade-off. Our contributions are:

[1] `Anonymous Code: https://anonymous.4open.science/r/D569/`

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.

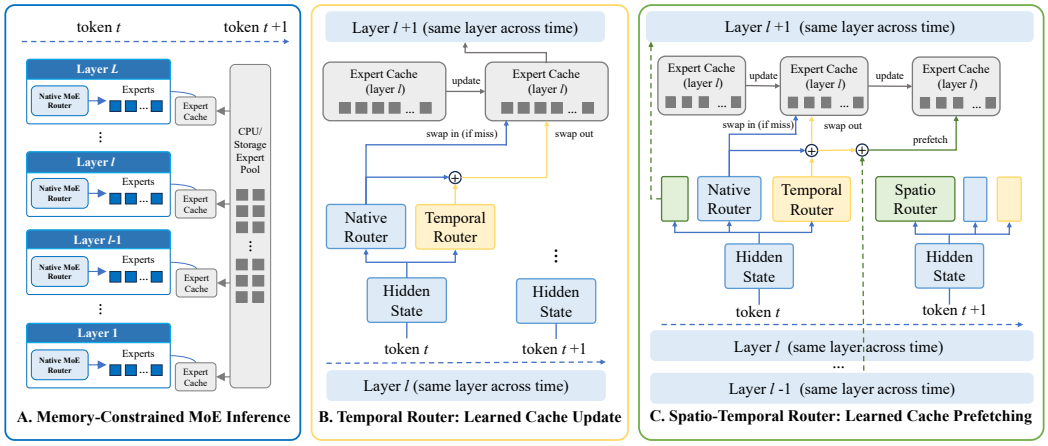


Figure 1: Overview of cache-aware MoE inference. (A) Each layer maintains a bounded GPU cache backed by a host/storage expert pool. (B) Temporal Router retains experts after access for later tokens at the same layer. (C) Spatio-Temporal Router adds causal pre-access refinement, followed by the same temporal update.

- A model-side cache-aware post-training formulation that jointly adapts native routing behavior and auxiliary cache priorities without changing the inference-time Top-K rule.
- A stackable two-stage algorithm with independently deployable Temporal Router retention and optional causal Spatio Router refinement.
- Evaluation on two backbones and three benchmarks, including adaptation, component, capacity, refinement-budget, and loss-weight ablations under proactive-load-aware accounting.

2 Problem Formulation

We consider autoregressive decoding with L MoE layers and N routed experts per layer. At token t and layer l, the native router maps hidden state $h_{t,l}$ to

$$p^R_{t,l} = \mathrm{softmax}(W^R_l h_{t,l}) \in \mathbb{R}^N, \qquad S_{t,l} = \mathrm{TopK}(p^R_{t,l}).$$

Each layer maintains a cache $\mathcal{C}_{t,l} \subseteq \{1, \ldots, N\}$ with $|\mathcal{C}_{t,l}| = B$ after initialization, where $B \geq K$. Here, $\mathcal{C}_{t,l}$ denotes the resident experts before access. Every selected expert in $S_{t,l} \setminus \mathcal{C}_{t,l}$ must be demand-loaded before its computation can proceed. Auxiliary routers do not override $S_{t,l}$. Preserving native routing means retaining the Top-K rule, not freezing its parameters.

Cache decisions occur at two points. *Post-access retention* keeps experts for later tokens and cannot prevent misses at the completed access. *Pre-access refinement* uses causal information to improve residency before a target access. Temporal Router uses retention only; Spatio-Temporal Router adds refinement before the same retention stage. We optimize the resulting transfer demand while tracking task quality, independently of a particular runtime scheduler.

3 Method

3.1 Unified Cache-Routing Framework

Figure 1 shows the cache lifecycle; Appendix C compares the architectures. The update-only mode accesses the temporal carry-over cache $\mathcal{C}^T_{t,l}$ directly; the full mode refines it into $\widetilde{\mathcal{C}}^{ST}_{t,l}$. After native expert computation, both apply the same retention rule to obtain $\mathcal{C}^T_{t+1,l}$.

We combine normalized router distributions by equal-weight addition, without a learned fusion module. For priority $s \in \mathbb{R}^N$, let $\theta_B(s)$ be its B-th largest entry. A soft Top-B membership

Algorithm 1 Shared Post-Access Cache Update $\mathcal{U}_B$

function UPDATECACHE($\mathcal{C}, S, s, B$)
  $\mathcal{C}^+ \leftarrow \mathcal{C}$
  $\mathcal{D} \leftarrow S \setminus \mathcal{C}^+$ ▷ demand-loaded selected experts
  for all $e \in \mathcal{D}$ **do**
    if $|\mathcal{C}^+| < B$ **then**
      $\mathcal{C}^+ \leftarrow \mathcal{C}^+ \cup \{e\}$
    else
      $\mathcal{V} \leftarrow \mathcal{C}^+ \setminus S$ ▷ protect current selected experts
      $v \leftarrow \arg\min_{u \in \mathcal{V}} s_u$
      $\mathcal{C}^+ \leftarrow (\mathcal{C}^+ \setminus \{v\}) \cup \{e\}$
    end if
  end for
  return $\mathcal{C}^+$
end function

surrogate and the shared coverage loss are

$$m_e(s) = \sigma\left(\frac{s_e - \theta_B(s)}{\tau}\right), \qquad \ell_{\text{cache}}(s, q) = \sum_{e=1}^{N} q_e\big(1 - m_e(s)\big).$$

Using the full native-router target distribution q, rather than hard Top-K labels, retains uncertainty near the selection boundary and supplies dense supervision. This soft objective learns relative priorities, not an unconstrained physical cache: the inference rules below determine which experts may enter and enforce the exact capacity. The two modes differ in their causal inputs, supervision targets, and when the resulting priority is applied. Retention is specified in Algorithm 1; full inference appears in Appendix A.

3.2 TEMPORAL ROUTER: POST-ACCESS RETENTION

Temporal Router predicts same-layer demand at the next token and combines it with current native routing:

$$p^T_{t,l} = \text{softmax}(W_l^T h_{t,l}), \qquad s^T_{t,l} = p^R_{t,l} + p^T_{t,l}.$$

The next-token native distribution supervises same-layer coverage through

$$\ell^T_{t,l} = \sum_{e=1}^{N} p^R_{t+1,l,e}\big(1 - m_e(s^T_{t,l})\big).$$

With Ω_T containing positions with a valid next-token target, training minimizes

$$\mathcal{L}_T = \frac{1}{|\Omega_T|} \sum_{(t,l) \in \Omega_T} \ell^T_{t,l}, \qquad \mathcal{L} = \mathcal{L}_{LM} + \lambda_T \mathcal{L}_T.$$

At inference, the native router accesses $\mathcal{C}^T_{t,l}$, demand-loads missing selected experts, and executes the layer. The post-access operator then forms

$$\mathcal{C}^T_{t+1,l} = \mathcal{U}_B\big(\mathcal{C}^T_{t,l}, S_{t,l}, s^T_{t,l}\big).$$

The operator inserts selected experts that were demand-loaded for the current computation, protects all experts in $S_{t,l}$, and evicts the lowest-priority non-selected residents when necessary. It does not admit arbitrary high-scoring experts from the external pool: every insertion is already available from the completed access. Temporal Router therefore learns retention without introducing proactive transfers. Algorithm 1 specifies this shared update.

3.3 SPATIO-TEMPORAL ROUTER: PRE-ACCESS REFINEMENT

The full mode adds a Spatio Router at source layer l, producing $p^S_{t,l} = \text{softmax}(W_l^S h_{t,l})$ for the next MoE position in causal execution order. For target (t, l), its predecessor is

$$\rho(t,l) = \begin{cases} (t-1, L), & l = 1, \\ (t, l-1), & l > 1. \end{cases}$$

Thus, the first layer uses the preceding token's final-layer prediction; other layers use the preceding layer of the same token. The pre-access priority combines temporal carry-over with this prediction:

$$a^T_{t,l} = p^R_{t-1,l} + p^T_{t-1,l}, \qquad s^{ST}_{t,l} = a^T_{t,l} + p^S_{\rho(t,l)}.$$

Equivalently, the first-layer wrap-around and within-token cases are

$$s^{ST}_{t,l} = \begin{cases} p^R_{t-1,1} + p^T_{t-1,1} + p^S_{t-1,L}, & l = 1, \\ p^R_{t-1,l} + p^T_{t-1,l} + p^S_{t,l-1}, & l > 1. \end{cases}$$

Each sum refers to experts at the target layer; the Spatio Router is indexed by the source that produces its prediction. Because this priority is available before target access, its training target is the current native distribution:

$$\ell^{ST}_{t,l} = \sum_{e=1}^{N} p^R_{t,l,e}\big(1 - m_e(s^{ST}_{t,l})\big).$$

Averaging over valid targets Ω_{ST} gives

$$\mathcal{L}_{ST} = \frac{1}{|\Omega_{ST}|} \sum_{(t,l)\in\Omega_{ST}} \ell^{ST}_{t,l}, \qquad \mathcal{L} = \mathcal{L}_{LM} + \lambda_{ST}\mathcal{L}_{ST}.$$

Here, Ω_{ST} requires both a previous-token same-layer state and a causal predecessor.

Before access, the full mode examines the top-R candidates under $s^{ST}_{t,l}$, in descending priority. A non-resident candidate replaces the lowest-priority resident only if its score is higher; each insertion is a proactive load. The native router then selects $S_{t,l}$, demand-loads remaining misses, and executes the layer. The shared temporal update produces $\mathcal{C}^T_{t+1,l}$ (Algorithm 2, Appendix A). The soft Top-B objective learns priorities, whereas the inference-only Top-R filter limits candidates rather than actual loads; R can be changed without retraining.

3.4 Training and Deployment Modes

Main runs jointly train the full backbone and auxiliary routers. Native distributions in the cache loss remain differentiable, so the objective can adapt expert demand as well as cache priorities. The full mode uses $\mathcal{L}_{ST}$, with neither a separate Temporal Router checkpoint nor an additional $\mathcal{L}_T$ term. Future/target distributions provide teacher-forced training supervision only; inference uses causal hidden states and stored outputs. The LM-only reference omits auxiliary routers and optimizes $\mathcal{L}_{LM}$ alone ($sw = 0$). The auxiliary-only control freezes the LM-only backbone and trains only the added routers. This separates auxiliary prediction on fixed model representations from joint adaptation; it does not isolate the contribution of any one backbone component.

Router initialization. The auxiliary routers are initialized by copying native-router weights, not by random initialization:

$$W^T_l \leftarrow W^R_l, \qquad W^S_l \leftarrow \begin{cases} W^R_{l+1}, & l < L, \\ W^R_1, & l = L. \end{cases}$$

Temporal Router starts from the same-layer router, while Spatio Router starts from its prediction target's router, including the final-to-first-layer wrap-around. Appendix B details evaluation controls.

4 Experiments

4.1 Experimental Setup

Backbone models. We use Qwen3-30B-A3B-Instruct-2507 (Team, 2025) and GPT-OSS-20B (Agarwal et al., 2025). Qwen3 has 48 MoE layers, approximately 30.5B total and 3.3B activated parameters, and 128 routed experts per layer with top-8 selection. GPT-OSS has 24 layers, approximately 21B total and 3.6B activated parameters, and 32 experts per layer with top-4 selection. Both use the same post-training and evaluation pipeline.

Method	Decision Availability	Model	Added Params.	GSM8K				MATH				CommonsenseQA			
				Acc.	Hit	Adj. Hit	Load	Acc.	Hit	Adj. Hit	Load	Acc.	Hit	Adj. Hit	Load
Cache-Update Methods															
MoE / LRU	Prev. token	Qwen3	–	**85.44**	61.19	–	1407	**58.22**	64.30	–	1294	**87.39**	56.44	–	1579
		GPT-OSS	–	61.87	65.58	–	1645	**43.74**	67.15	–	1569	84.68	65.10	–	1668
MoE / LFU	Prev. token	Qwen3	–	**85.44**	59.99	–	1450	**58.22**	63.00	–	1341	**87.39**	58.27	–	1512
		GPT-OSS	–	61.87	68.63	–	1499	**43.74**	65.37	–	1654	84.68	69.29	–	1467
MoE / LRFU	Prev. token	Qwen3	–	**85.44**	62.67	–	1353	**58.22**	64.29	–	1294	**87.39**	57.89	–	1526
		GPT-OSS	–	61.87	69.55	–	1455	**43.74**	68.66	–	1497	84.68	68.34	–	1513
Temporal Router	Prev. token	Qwen3	12.6M	**85.44**	**73.13**	–	**974**	57.82	**75.00**	–	**906**	86.16	**91.61**	–	**304**
		GPT-OSS	2.2M	**63.99**	**72.84**	–	**1298**	**43.74**	**71.84**	–	**1345**	**85.83**	**71.54**	–	**1360**
Prefetching Methods															
Least-Stale (SpecMD)	Prev. token	Qwen3	–	**85.44**	68.52	31.12	5496	**58.22**	70.02	31.55	5505	**87.39**	67.56	30.17	5667
		GPT-OSS	–	61.87	70.07	35.59	6058	**43.74**	72.29	37.03	5873	**84.68**	71.51	36.91	5839
ProMoE	Prev. layer	Qwen3	96.0M	**85.44**	89.38	64.38	1792	**58.22**	88.09	64.21	1779	**87.39**	85.38	60.71	2003
		GPT-OSS	48.0M	61.87	93.54	**67.94**	**2109**	**43.74**	91.16	**68.19**	**2032**	**84.68**	92.95	67.78	2111
FineMoE	Prev. layer	Qwen3	–	**85.44**	76.67	54.84	2288	**58.22**	71.45	51.41	2447	**87.39**	74.08	51.40	2538
		GPT-OSS	–	61.87	85.63	65.02	2201	**43.74**	71.96	50.40	3384	**84.68**	75.49	53.08	3188
Temporally-Extended MoE	Curr. layer	Qwen3	52.0M	81.50	**100.00**	52.34	3299	51.04	**100.00**	49.58	3685	84.36	**100.00**	58.77	2543
		GPT-OSS	23.0M	60.27	**100.00**	52.78	4275	43.22	**100.00**	57.40	3546	64.29	**100.00**	62.61	2853
Spatio-Temporal Router	Prev. layer	Qwen3	25.2M	83.40	90.62	**69.03**	**1474**	57.66	88.40	**65.36**	**1698**	84.11	95.56	**78.74**	**935**
		GPT-OSS	4.4M	**64.52**	85.68	58.81	2951	42.46	89.12	63.53	2625	84.60	90.26	**71.30**	**1736**

Table 1: Main results on Qwen3 and GPT-OSS. LRU/LFU/LRFU use matched LM-only backbones ($sw = 0$). Decision Availability denotes when all decision inputs become available: Prev. token permits a longer potential overlap window than Prev. layer; Curr. layer has no lookahead. Our full mode includes the last-to-first-layer token wrap-around. Added Params. counts new inference-time parameters, excluding external stores. Accuracy and hit rates are percentages; Load is MB/decode token. Results are five-seed means; bold/underline denote best/second-best values within each method category, backbone, task, and metric.

Datasets. GSM8K contains 7,473 training and 1,319 test grade-school mathematics problems (Cobbe et al., 2021). MATH contains 7,500 training and 5,000 test competition-level problems (Hendrycks et al., 2021). CommonsenseQA contains 9,741 training, 1,221 validation, and 1,140 hidden-test questions (Talmor et al., 2019); we evaluate on validation because test labels are not public. Accuracy uses exact match for GSM8K and CommonsenseQA and symbolic or numeric matching for MATH.

Baselines and protocol. We compare LRU, LFU, and LRFU for cache update, and Least-Stale (SpecMD), ProMoE, FineMoE, and Temporally Extended MoE for prefetching (Section 6). Classical policies run on matched LM-only post-trained backbones using the same data, optimizer, and schedule; learned baselines follow their prescribed adaptation. Thus, we compare complete inference configurations, not policies on an identical routing trace. All methods share cache-accounting rules that charge every proactive insertion.

Main settings (B, τ, sw, R) are $(20, 0.03, 0.1, 15)$ for Qwen3 and $(8, 0.05, 0.01, 6)$ for GPT-OSS, where sw denotes the relevant cache-loss weight and R applies only to the full mode. These settings were fixed before task evaluation and shared across datasets. Tables 1 and 2 report means over the same five distinct, fixed seeds.

Training and decoding. Qwen3 and GPT-OSS use learning rates of 10^{-5} and 5×10^{-6}, respectively, for four post-training epochs. We use fused AdamW with cosine decay and 3% warmup, training on the full training splits without validation-based checkpoint selection. Main runs update the backbone and auxiliary routers; LM-only controls use the same schedule without the cache loss or auxiliary routers. Training uses bfloat16, TF32, and gradient checkpointing, without LoRA or weight quantization. Experiments run on an eight-accelerator node with 140 GB of device memory per accelerator. We use weight decay 0.1, gradient clipping at 1.0, per-device batch size 1, and eight gradient-accumulation steps. Maximum sequence lengths are 512 for GSM8K and CommonsenseQA and 2048 for MATH. No cache-specific load-balancing loss is added. Evaluation uses each backbone's chat template and task-specific system instruction, with greedy decoding and the

KV cache. Generation stops at EOS or a budget of 512, 1,024, or 64 new tokens for GSM8K, MATH, or CommonsenseQA, respectively. Additional evaluation and ablation details are in Appendix B.

Baseline settings. LRFU uses $\lambda = 0.5$. SpecMD/Least-Stale uses lookahead 3 and a prefetch budget equal to cache capacity. ProMoE uses training-split traces and lookahead 3. FineMoE uses a 1K-entry expert-map store built from training traces, lookahead 3, a cache-sized prefetch budget, 64 candidates, and retrieval top-k values of 8 for Qwen3 and 4 for GPT-OSS. Temporally Extended MoE uses a cache-sized option set, 128-dimensional controller and set embeddings, copy-router initialization, and its original losses.

Metrics. We report task accuracy, hard hit rate, load-adjusted hit rate, and expert-weight traffic. A trace-driven simulator synchronized with decoding counts routed expert accesses A, demand misses D, proactive loads P, and decode tokens T; shared experts are excluded. Each miss or insertion costs one complete expert transfer of size S_{exp} MB:

$$\text{Hit} = 1 - \frac{D}{A}, \qquad \text{AdjHit} = \frac{A - D}{A + P}, \qquad \text{Load/token} = \frac{(D + P)S_{\text{exp}}}{T}.$$

Prefill initializes the decode cache, but its transfers are excluded. For update-only methods, $P = 0$ and AdjHit equals Hit, so it is omitted. Load/token is the primary algorithmic cost; AdjHit provides a normalized penalty for proactive traffic. These metrics measure decode-stage transfer demand, not end-to-end latency or transfer–computation overlap.

4.2 Main Results

Update-only retention. Table 1 shows that Temporal Router achieves the highest hit rate and lowest Load among update-only configurations on all six backbone–task pairs. On Qwen3, gains over the strongest classical hit-rate baseline are 10.46, 10.70, and 33.34 points on GSM8K, MATH, and CommonsenseQA; Load falls from 1353 to 974, 1294 to 906, and 1512 to 304 MB/token. GPT-OSS shows the same ranking with 2.2M added inference-time parameters.

Prefetch-enabled refinement. Among evaluated prefetchers, Spatio-Temporal Router leads all Qwen3 tasks in adjusted hit rate and Load. Relative to ProMoE, it improves adjusted hit by 1.15–18.03 points and reduces traffic by 4.6–53.3%. GPT-OSS results are mixed: the full mode leads CommonsenseQA and ranks second on MATH in both metrics; on GSM8K, it attains the highest accuracy but weaker traffic efficiency. The two operating modes are not interchangeable: Temporal Router has lower total Load in all six settings, while the full mode achieves higher pre-access coverage by spending proactive traffic.

Quality and overhead. The full mode adds 25.2M parameters on Qwen3 and 4.4M on GPT-OSS (0.083% and 0.021% inference-time model-size overhead), versus 96.0M and 48.0M for ProMoE. FineMoE adds no trainable predictor but uses an external expert-map store. Task quality must be read alongside traffic: on Qwen3, full-mode accuracies are 83.40%, 57.66%, and 84.11%, compared with LM-only values of 85.44%, 58.22%, and 87.39%. Thus, better residency does not imply unchanged task behavior. Raw hit alone is also insufficient: Temporally Extended MoE attains 100% hit by restricting routing to its option set, yet has less favorable traffic and accuracy.

5 Analysis and Discussion

Adaptation scope. Table 2 compares auxiliary-only and joint adaptation. At $sw = 0.1$, auxiliary-only training preserves accuracy: Temporal Router reaches 63.01% hit at 1340.71 MB/token; the full mode reaches 67.44% hit and 62.53% adjusted hit at 1465.59 MB/token. Joint training raises Temporal Router to 73.13% hit at 973.86 MB/token without accuracy loss. The full mode reaches 90.62% hit and 69.03% adjusted hit at 1474.00 MB/token, with a 2.04-point accuracy drop. Relative to auxiliary-only, its gain is higher coverage, not lower traffic (+8.41 MB/token).

Router composition. Temporal-only repeats the standard Temporal Router at $sw = 0.1$. Spatio-only removes p^T, trains with $s^S_{t,l} = p^R_{t-1,l} + p^S_{\rho(t,l)}$, and retains cross-layer prefetching. Demand-loaded experts enter the cache with LRU eviction rather than learned temporal retention. Relative to

Temporal Router				Spatio-Temporal Router				
Setting	**Acc.**	**Hit**	**Load**	**Setting**	**Acc.**	**Hit**	**Adj. Hit**	**Load**
MoE / LRU	85.44	61.19	1406.50	MoE / LRU	85.44	61.19	–	1406.50
$sw = 0.1$ auxiliary only	85.44 (=0.00)	63.01 (↑1.82)	1340.71 (↓65.79)	$sw = 0.1$ auxiliary only	85.44 (=0.00)	67.44 (↑6.25)	62.53 (↑1.34)	1465.59 (↑59.09)
$sw = 0.1$ temporal only	85.44 (=0.00)	73.13 (↑11.94)	973.86 (↓432.64)	$sw = 0.1$ spatio only	83.24 (↓2.20)	93.04 (↑31.85)	66.94 (↑5.75)	1665.51 (↑259.01)
$sw = 0.1$	85.44 (=0.00)	73.13 (↑11.94)	973.86 (↓432.64)	$sw = 0.1$	83.40 (↓2.04)	90.62 (↑29.43)	69.03 (↑7.84)	1474.00 (↑67.50)
$sw = 0.3$	84.99 (↓0.45)	79.02 (↑17.83)	760.30 (↓646.20)	$sw = 0.3$	83.47 (↓1.97)	93.67 (↑32.48)	73.42 (↑12.23)	1229.04 (↓177.46)
$sw = 0.5$	82.56 (↓2.88)	81.98 (↑20.79)	653.07 (↓753.43)	$sw = 0.5$	81.80 (↓3.64)	95.01 (↑33.82)	75.66 (↑14.47)	1107.47 (↓299.03)
$sw = 1.0$	79.15 (↓6.29)	85.31 (↑24.12)	532.33 (↓874.17)	$sw = 1.0$	81.35 (↓4.09)	96.34 (↑35.15)	78.00 (↑16.81)	985.16 (↓421.34)
$sw = 2.0$	74.60 (↓10.84)	88.44 (↑27.25)	418.89 (↓987.61)	$sw = 2.0$	76.50 (↓8.94)	97.41 (↑36.22)	79.67 (↑18.48)	900.91 (↓505.59)

Table 2: Qwen3/GSM8K ablations. MoE/LRU is the LM-only reference ($sw = 0$); auxiliary-only freezes this backbone. Temporal-only repeats the standard Temporal Router; Spatio-only combines prefetching with LRU eviction. Other sw rows jointly train the backbone and auxiliary routers. Accuracy/hit rates are percentages; Load is MB/token. Five-seed means are shown, with changes from MoE/LRU in parentheses. Red marks higher hit or lower traffic; blue marks lower accuracy or higher traffic.

B	Temporal Router			Spatio-Temporal Router			
	Acc.	**Hit**	**Load**	**Acc.**	**Low** R Hit / Adj. / Load	**Mid** R Hit / Adj. / Load	**Full** $R = B$ Hit / Adj. / Load
12	84.46	62.91	1344	85.44	$R = 4$ 66.53 / 63.56 / 1382	$R = 8$ 73.92 / 65.12 / 1435	$R = 12$ 80.79 / 60.63 / 1901
20	85.44	73.13	974	83.40	$R = 8$ 81.84 / 74.51 / 1015	$R = 15$ 90.62 / 69.03 / 1474	$R = 20$ 93.49 / 57.93 / 2461
30	85.37	80.00	725	83.85	$R = 8$ 86.14 / 80.40 / 761	$R = 20$ 95.11 / 71.14 / 1398	$R = 30$ 97.23 / 53.39 / 3077

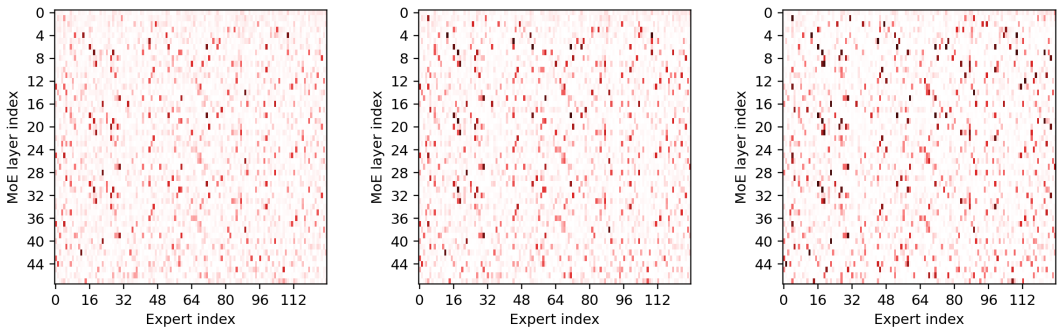
Table 3: Cache capacity B and refinement budget R on Qwen3/GSM8K; the main setting is $B = 20, R = 15$. Spatio-Temporal cells list R above Hit/Adj. Hit/Load. Budgets at each B replay one checkpoint with shared accuracy. Metrics are percentages except Load (MB/token).

the full mode, hard hit rises (93.04% versus 90.62%), but adjusted hit falls (66.94% versus 69.03%) and Load increases (1665.51 versus 1474.00 MB/token). The combined retention–refinement configuration thus uses less traffic than LRU-backed Spatio-only, despite lower immediate coverage.

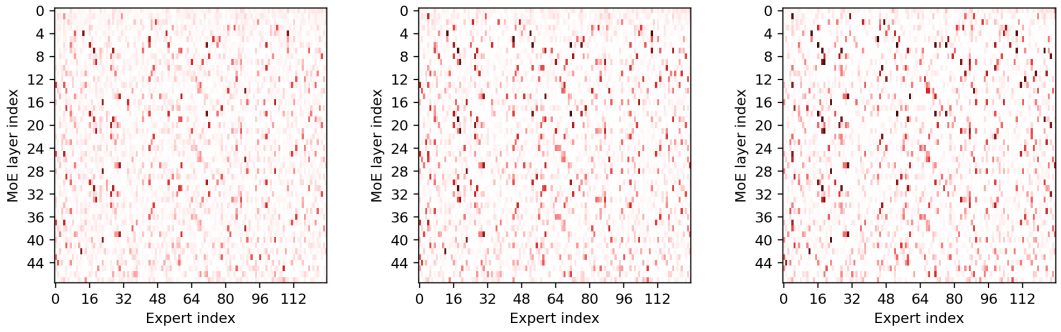
Proactive traffic. Higher raw hit need not imply lower total Load, since every prefetch transfer is counted. With fixed traces and equal transfer costs, exposed loading ranges from demand-only to serialized demand-plus-prefetch cost. Hit and AdjHit measure coverage, not latency bounds; Appendix B.3 gives the assumptions.

Cache capacity and refinement budget. Table 3 studies B and R on Qwen3/GSM8K. For Temporal Router, increasing B from 12 to 30 raises hit from 62.91% to 80.00% and reduces Load from 1344 to 725 MB/token, with accuracy within one point. At common $R = 8$, the full mode's hard/adjusted hit rises from 73.92%/65.12% to 86.14%/80.40%, and Load falls from 1435 to 761 MB/token. Because each B uses a separate checkpoint, its effects include model adaptation.

Within each B, replaying the same checkpoint isolates inference-time R. At $B = 20$, raising R from 8 to 15 and 20 increases hit from 81.84% to 90.62% and 93.49%, but decreases adjusted hit from

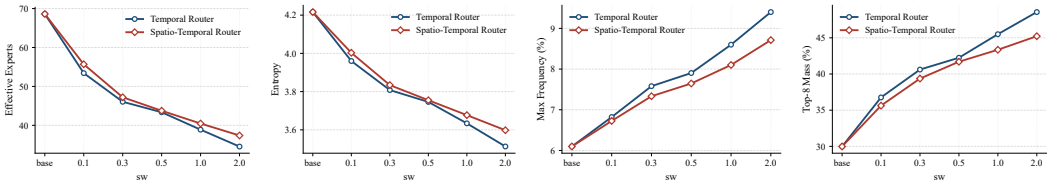


(a) Temporal, $sw = 0.1$ **(b)** Temporal, $sw = 0.5$ **(c)** Temporal, $sw = 2.0$



(d) Spatio-Temporal, $sw = 0.1$ **(e)** Spatio-Temporal, $sw = 0.5$ **(f)** Spatio-Temporal, $sw = 2.0$

Figure 2: Native expert-selection frequency on Qwen3/GSM8K. Rows show Temporal Router (top) and Spatio-Temporal Router (bottom); darker cells indicate more frequent selection.



(a) Effective experts (b) Entropy (c) Max frequency (d) Top-8 mass

Figure 3: Expert-use concentration versus cache-loss weight, averaged over MoE layers. Base is the LM-only reference. Per-layer effective-expert count is $\exp(\text{entropy})$; max frequency and top-8 mass measure the most-used expert's frequency and the eight most-used experts' total frequency.

74.51% to 69.03% and 57.93% and raises Load from 1015 to 1474 and 2461 MB/token. Across all three capacities, $R = B$ yields the highest raw hit but worst traffic efficiency. The main $R = 15$ setting favors coverage, while smaller budgets offer lower-traffic operating points without retraining.

Cache-loss weight. Table 2 varies sw on Qwen3/GSM8K as a post hoc sensitivity study, not task-specific tuning of the main settings. For Temporal Router, increasing sw from 0.1 to 2.0 raises hit from 73.13% to 88.44% and lowers Load from 973.86 to 418.89 MB/token. Relative to MoE/LRU, these settings improve hit by 11.94–27.25 points and reduce traffic by 432.64–987.61 MB/token. The accuracy cost grows: $sw = 0.1$ preserves 85.44% accuracy, $sw = 0.3$ loses 0.45 points, and $sw = 2.0$ loses 10.84 points.

For the full mode, stronger supervision raises hard hit from 90.62% to 97.41% and adjusted hit from 69.03% to 79.67%. Demand-traffic savings outweigh proactive insertions from $sw = 0.3$ onward: $sw = 0.1$ increases baseline Load by 67.50 MB/token, while $sw = 0.3$ and $sw = 2.0$ reduce it by 177.46 and 505.59 MB/token. Accuracy losses are 1.97–2.04 points at the two smaller weights and

8.94 points at $sw = 2.0$. Moderate supervision therefore offers a useful quality–traffic trade-off, whereas large weights over-optimize cache locality at the expense of task quality.

Expert-use concentration. Figure 2 shows increasingly concentrated native expert use as sw grows. Auxiliary routers do not override the native Top-K rule, but joint post-training can reshape the distribution on which that rule operates. The observed concentration is consistent with improved cache locality, although it does not by itself establish greater temporal stability.

Figure 3 quantifies this trend. For Temporal Router, increasing sw from 0.1 to 2.0 raises maximum expert frequency from 6.82% to 9.40% and top-8 mass from 36.76% to 48.56%, while entropy falls from 3.96 to 3.51 and effective experts from 53.41 to 34.54. The full mode changes in the same direction with flatter curves: maximum frequency rises from 6.73% to 8.71%, and top-8 mass from 35.63% to 45.22%. Together with the auxiliary-only control, these observations support joint adaptation toward more cache-compatible demand. They characterize aggregate model behavior, not individual-router contributions. Moderate supervision retains substantial routing diversity; excessive concentration accompanies the accuracy losses at large sw.

6 Related Work

LLM and MoE serving. FlexGen (Sheng et al., 2023) and vLLM/PagedAttention (Kwon et al., 2023) address memory pressure through tensor placement, KV-cache management, and batching. MoE serving additionally manages large, sparse, dynamically selected expert weights. DeepSpeed-MoE (Rajbhandari et al., 2022) and Tutel (Hwang et al., 2023) optimize parallel execution, while MoE-Infinity (Xue et al., 2024) targets offloading through tracing, caching, and prefetching. Related work studies serverless placement and speculative scheduling (Yu et al., 2026b; Li et al., 2025). Unlike these serving systems, we learn model-side cache priorities.

Expert caching and prefetching. LRU and LFU use recency and frequency (Mattson et al., 1970; Robinson & Devarakonda, 1990); LRFU interpolates between them (Lee et al., 2001). MoE-specific methods exploit richer signals: Least-Stale uses access history (Hoang et al., 2026), ProMoE trains an activation-based predictor (Song et al., 2024), FineMoE retrieves expert maps (Yu et al., 2026a), and Temporally Extended MoE learns whether to retain or switch expert sets (Shen & Henderson, 2026). We instead learn cache priorities jointly with model adaptation.

Spatio-temporal prefetching. STEP uses adaptive spatio-temporal expert prefetching (Liu et al., 2026); ST-MoE combines profiled cross-layer and consecutive-token correlations with table-based prediction and reconfigurable hardware (Zhao et al., 2026). Both emphasize inference-time prediction or system support. We focus on cache-aware model adaptation and support both update-only retention and bounded pre-access refinement. Their reported latency reflects integrated runtime or hardware pipelines, so we compare designs qualitatively rather than treating those measurements as directly comparable to our traffic metric.

7 Conclusion

We presented cache-aware post-training for expert-cache management in memory-constrained MoE inference. Temporal Router learns same-layer retention without proactive loading; Spatio-Temporal Router adds bounded causal refinement while preserving the native inference-time Top-K rule. Across two backbones and three tasks, Temporal Router consistently reduces transfer demand relative to matched LM-only replacement configurations. The full mode offers stronger pre-access coverage, with favorable traffic results on Qwen3 and task-dependent results on GPT-OSS. Auxiliary-only and component ablations support the value of joint model adaptation and temporal retention, while the budget sweeps distinguish the roles of cache capacity, refinement aggressiveness, and cache-loss weight. These controls expose a quality–coverage–traffic trade-off rather than a single universally optimal configuration. The results characterize algorithmic transfer demand; end-to-end performance remains dependent on the runtime, as discussed in Appendix D.

Ethics Statement

This work studies algorithmic expert-cache management using public models and benchmarks, without human-subject studies or high-stakes deployment. Preserving native Top-K selection does not guarantee unchanged behavior after post-training. Downstream risks depend on the underlying language models and deployment settings; efficiency gains are not evidence of safety.

Reproducibility Statement

Sections 3.1–3.4 specify the framework and objectives. Algorithm 1 and Appendix A specify inference. Section 4 and Appendix B give training, decoding, initialization, baseline, and evaluation settings. Source code and configurations are provided in the supplementary material.

AI Use Statement

Generative AI tools assisted with manuscript organization and editing, literature search and summarization, and preliminary figure design. The authors verified the implementation, measurements, and scientific claims, reviewed all AI-assisted content, and take responsibility for the manuscript.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Duc Hoang, Ajay Jaiswal, Mohammad Samragh, and Minsik Cho. Specmd: A comprehensive study on speculative expert prefetching. *arXiv preprint arXiv:2602.03921*, 2026.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, et al. Tutel: Adaptive mixture-of-experts at scale. *Proceedings of Machine Learning and Systems*, 5:269–287, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. Lrfu: A spectrum of policies that subsumes the least recently used and least frequently used policies. *IEEE transactions on Computers*, 50(12):1352–1361, 2001.

Yan Li, Pengfei Zheng, Shuang Chen, Zewei Xu, Yuanhao Lai, Yunfei Du, and Zhengang Wang. Speculative moe: Communication efficient parallel moe inference with speculative token and expert pre-scheduling. *arXiv preprint arXiv:2503.04398*, 2025.

Fangxin Liu, Ning Yang, Zongwu Wang, Chenyang Guan, Haomin Li, Yu Feng, Liqiang Lu, Xiang Li, Siran Yang, Jiamang Wang, Lin Qu, Li Jiang, and Haibing Guan. Step: Adaptive spatio-temporal expert prefetching for low-latency and memory-efficient moe inference. In *2026 ACM/IEEE 53rd Annual International Symposium on Computer Architecture (ISCA)*, pp. 1336–1350, 2026.

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

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale. In *International conference on machine learning*, pp. 18332–18346. PMLR, 2022.

John T Robinson and Murthy V Devarakonda. Data cache management using frequency-based replacement. In *Proceedings of the 1990 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pp. 134–142, 1990.

Zeyu Shen and Peter Henderson. Temporally extended mixture-of-experts models. *arXiv preprint arXiv:2604.20156*, 2026.

Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. Flexgen: High-throughput generative inference of large language models with a single gpu. In *International Conference on Machine Learning*, pp. 31094–31116. PMLR, 2023.

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

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4149–4158, 2019.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. Moe-infinity: Offloading-efficient moe model serving. *arXiv e-prints*, pp. arXiv–2401, 2024.

Hanfei Yu, Xingqi Cui, Hong Zhang, Hao Wang, and Hao Wang. Taming latency-memory trade-off in moe-based LLM serving via fine-grained expert offloading. In Antonio Barbalace, Luo Mai, Roxana Geambasu, and Peter R. Pietzuch (eds.), *Proceedings of the 21st European Conference on Computer Systems, EuroSys 2026, McEwan Hall/The University of Edinburgh, Edinburgh, Scotland, UK, April 27-30, 2026*, pp. 176–191. ACM, 2026a.

Hanfei Yu, Bei Ouyang, Shwai He, Ang Li, and Hao Wang. Moeless: Efficient moe llm serving via serverless computing. *arXiv preprint arXiv:2603.06350*, 2026b.

Yingnan Zhao, Razvan Bunescu, Ahmed Louri, Avinash Karanth, and Ke Wang. A spatio-temporal expert prefetching framework for efficient moe-based llm inference, 2026.

A Full Inference Algorithms

Algorithm 2 gives the full prefetch-enabled procedure and invokes the shared retention operator in Algorithm 1. Temporal Router admits only experts already resident or demand-loaded for the current computation; the full mode additionally charges each proactive insertion before target access.

Algorithm 2 Full Spatio-Temporal Router Inference at (t, l)

Require: $\mathcal{C}^T_{t,l}$; $p^R_{t-1,l}, p^T_{t-1,l}, p^S_{\rho(t,l)}$; budgets R, B

$a^T_{t,l} \leftarrow p^R_{t-1,l} + p^T_{t-1,l}$
$s^{ST}_{t,l} \leftarrow a^T_{t,l} + p^S_{\rho(t,l)}$
$\mathcal{A} \leftarrow \mathrm{Top}_R(s^{ST}_{t,l})$
$\widetilde{\mathcal{C}}^{ST}_{t,l} \leftarrow \mathcal{C}^T_{t,l}$
for all $e \in \mathcal{A}$ in descending order of $s^{ST}_{t,l,e}$ **do**
    if $e \notin \widetilde{\mathcal{C}}^{ST}_{t,l}$ **then**
        if $|\widetilde{\mathcal{C}}^{ST}_{t,l}| < B$ **then**
            $\widetilde{\mathcal{C}}^{ST}_{t,l} \leftarrow \widetilde{\mathcal{C}}^{ST}_{t,l} \cup \{e\}$ ▷ proactive load
        else
            $v \leftarrow \arg\min_{u \in \widetilde{\mathcal{C}}^{ST}_{t,l}} s^{ST}_{t,l,u}$
            if $s^{ST}_{t,l,e} > s^{ST}_{t,l,v}$ **then**
                $\widetilde{\mathcal{C}}^{ST}_{t,l} \leftarrow (\widetilde{\mathcal{C}}^{ST}_{t,l} \setminus \{v\}) \cup \{e\}$ ▷ proactive load
            end if
        end if
    end if
end for
Compute $p^R_{t,l}$, $S_{t,l} = \mathrm{TopK}(p^R_{t,l})$, and $p^T_{t,l}$ from $h_{t,l}$
Demand-load $S_{t,l} \setminus \widetilde{\mathcal{C}}^{ST}_{t,l}$ and execute the MoE layer
$s^T_{t,l} \leftarrow p^R_{t,l} + p^T_{t,l}$
$\mathcal{C}^T_{t+1,l} \leftarrow \textsc{UpdateCache}(\widetilde{\mathcal{C}}^{ST}_{t,l}, S_{t,l}, s^T_{t,l}, B)$
return $\mathcal{C}^T_{t+1,l}$

B Implementation and Evaluation Details

B.1 Initialization and evaluation controls

The copy-initialization rules in Section 3.4 apply before cache-aware training. For Tables 1 and 2, every configuration runs once under each of the same five distinct, fixed seeds. Metrics are computed per run and then averaged across the five runs. The main refinement budgets are $R = 15$ for Qwen3 and $R = 6$ for GPT-OSS, both 75% of cache capacity. As described in the main text, R bounds the candidate set rather than the number of actual loads. The post-prefill cache initializes decoding, while all reported hit-rate and traffic statistics exclude prefill transfers. Each demand miss and proactive insertion is charged as one complete expert-weight transfer, without modeling scheduling or overlap.

B.2 Ablation and sensitivity configurations

Auxiliary-only training freezes the LM-only backbone, including W^R, and trains only W^T, or W^T and W^S. Temporal-only is the standard update-only mode, repeated in Table 2 rather than trained separately. Spatio-only removes Temporal Router and its learned cross-token retention stage. Its cache-coverage objective uses $s^S_{t,l} = p^R_{t-1,l} + p^S_{\rho(t,l)}$, and causal cross-layer prefetching continues with the same candidate-selection and score-based pre-access refinement rule. Any selected expert absent after prefetching is demand-loaded and inserted into the cache, with LRU evictions to maintain capacity. In the full mode, the corresponding demand-miss replacement uses Temporal Router

priorities instead. Spatio-only therefore removes learned temporal retention, not demand-miss insertion or all cache replacement.

Table 3 uses separately trained Qwen3/GSM8K checkpoints for $B \in \{12, 20, 30\}$. Each Spatio-Temporal checkpoint is replayed with its three listed inference-time budgets without retraining, so accuracy is shared within each B. The parameter R corresponds to `replace_lowest`; it is distinct from the native Top-K expert count. Both budget and loss-weight sweeps are post hoc sensitivity studies, not task-specific selection of the main hyperparameters.

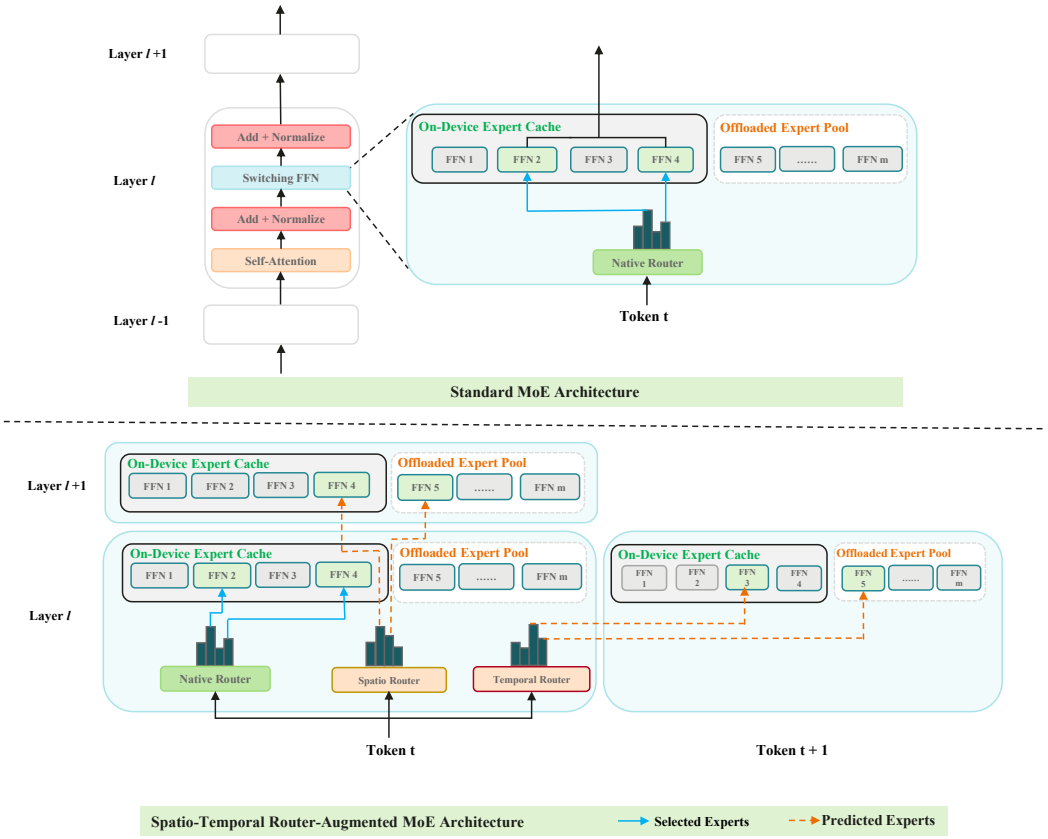


Figure 4: Standard MoE (top) and Spatio-Temporal Router-augmented MoE (bottom), with on-device expert caches and offloaded pools. Solid blue arrows denote native selection; dashed orange arrows denote predicted demand, not transfers. Temporal Router retains only resident or demand-loaded experts, even when predictions include offloaded experts. Proactive loading occurs only during bounded pre-access refinement; expert indices and cache contents are illustrative.

B.3 Interpreting asynchronous transfer overlap

Total transferred bytes do not depend on whether transfers overlap computation. To illustrate how overlap can affect exposed loading cost, fix the routing and cache trace, including A, D, P, T, and assume each expert transfer takes c_{exp} seconds. Let $\eta \in [0, 1]$ denote the fraction of proactive transfer time hidden by useful computation. Treating demand transfers as fully exposed and ignoring transfer contention and auxiliary overhead, the modeled expert-loading cost per decode token is

$$t_{\text{load}}(\eta) = \frac{c_{\text{exp}}}{T}\big[D + (1-\eta)P\big].$$

Complete prefetch overlap ($\eta = 1$) leaves only demand-transfer cost; no overlap ($\eta = 0$) charges both demand and proactive transfers. Accordingly,

$$\frac{Dc_{\text{exp}}}{T} \le t_{\text{load}}(\eta) \le \frac{(D+P)c_{\text{exp}}}{T}.$$

These two endpoints can be written in terms of the reported metric definitions:

$$t_{\text{load}}(1) = \frac{Ac_{\text{exp}}}{T}(1 - \text{Hit}),$$
$$t_{\text{load}}(0) = \frac{(A+P)c_{\text{exp}}}{T}(1 - \text{AdjHit}).$$

The normalizers differ: Hit discounts demand misses over A accesses, whereas AdjHit charges all transfers over $A+P$. Their numerical values therefore cannot be interpreted directly as the endpoints of a latency or speedup interval. Under the stated assumptions, a prefetch-enabled configuration can have higher total traffic yet lower exposed loading cost when enough proactive transfer time is hidden. The realized overlap fraction is not measured here, and perfect overlap may be infeasible because of limited lookahead or bandwidth. These bounds apply only to the simplified fixed-trace loading model, not to end-to-end latency; scheduling, contention, late prefetches, and auxiliary computation require runtime evaluation.

C Detailed Architectural Comparison

Figure 4 compares the standard MoE block with its router-augmented counterpart. The native, Temporal, and Spatio Routers share the layer input hidden state. Native routing selects experts for the current computation; Temporal Router scores retention for the next token at the same layer, while Spatio Router predicts demand at the next MoE position. The update-only mode retains Temporal Router and omits Spatio Router.

D Limitations

Our evaluation isolates algorithmic cache management rather than an end-to-end serving stack. Load/token measures simulated decode-stage expert-weight traffic; realized latency and throughput also depend on scheduling, transfer–computation overlap, interconnect bandwidth, and batching. Prefill initializes the cache but its transfers are excluded, so results characterize decode-stage rather than request-level efficiency. They establish neither wall-clock speedup nor energy savings.

Main runs require full-model post-training despite small inference-time parameter overhead. The auxiliary-only control supports joint adaptation but does not isolate changes in native-router parameters from other backbone changes. The full benchmark matrix uses one main cache capacity per backbone; capacity and refinement sensitivity are studied on Qwen3/GSM8K. Main comparisons and ablations use five seeds; the budget study replays fixed checkpoints.

Experiments focus on academic reasoning benchmarks and pure MoE models. Broader workloads, memory budgets, pretraining settings, and hybrid dense–sparse architectures remain untested. More concentrated native routing may improve locality but impair expert-parallel load balance. Finally, baseline algorithms are reproduced under common cache accounting rather than their original runtime environments; rankings may change under other workloads, memory hierarchies, or schedulers.