TEMPORAL MIXTURE-OF-EXPERTS: SERVING MEMORY THAT SCALES WITH ACTIVE PARAMETERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse Mixture-of-Experts (MoE) models activate only a small fraction of their parameters per token, yet serving one still requires the whole expert pool in fast memory, making inference impractical on memory-constrained local devices. We introduce *Temporal MoE*, a new routing policy that reuses all but one of the previous token's experts for each layer. This allows us to keep only the active experts in RAM instead of the entire pool, and schedule loading the incoming expert from flash to overlap with computing the reused ones, effectively hiding transfer overhead. When pre-trained from scratch, Temporal MoE reduces serving memory by $5.1\times$ while preserving 72–82% of the standard MoE advantage over dense models in cross-entropy and downstream accuracy. This scaling trend holds consistently across compute-optimal budgets from 10^{16} to 10^{19} FLOPs. Probing routers demonstrate that the temporal routers focus more on context rather than token identity and spread each expert's use more evenly across the token stream. In real-device hardware benchmarks across an RTX A6000, an Intel laptop, and a Pixel 10a where the model otherwise does not fit, an 11B-scale model keeps 66–83% of the standard MoE's decode speed, and the workstation realizes the $5.1\times$ reduction. Applying this constraint post-hoc to released large instruct MoEs demonstrates remarkable robustness (holding especially well for sparser models). Moreover, just two GPU-hours of distillation recovers most of the accuracy lost under the constraint for Gemma4-26B and Qwen3.5-35B. These results show that Temporal MoE can move the quality–memory frontier for local inference by making expert reuse a core part of both training and serving.

1 INTRODUCTION

Sparse MoE layers draw on a large parameter pool while computing only the k experts active per token (Shazeer et al., 2017; Dai et al., 2024; Jiang et al., 2024). At serving time these models still pay to keep every parameter in memory. In Qwen3.5-35B-A3B, the experts hold about 92% of all weights (Qwen Team, 2026), yet every expert must sit in fast memory even though most do nothing for a given token. Offloading systems attempt to stream experts from slower storage on demand (Eliseev & Mazur, 2023; Skliar et al., 2025), but per-token offload is bandwidth-infeasible. Decoding is memory-bound, so when flash is 15–100$\times$ slower than RAM, masking one expert's loading time requires the computation of 15–100 experts in the faster memory, but a standard MoE layer has only $k \approx 8$ active experts per token. Caching reduces how often experts must be loaded, but each load remains expensive. Standard routers also offer little reuse, since consecutive tokens share only about 20% of their expert choices (Jiang et al., 2024; Muennighoff et al., 2025; Liang et al., 2026).

We therefore design and train the model for the serving constraint itself. Our temporal router reuses all but one of the previous token's experts for each layer. This lets us keep only the k active experts of each layer resident in memory, instead of all the experts, because we can schedule the single new expert's weights to load from slow storage while the experts already in memory are computed, hiding the transfer overhead. However, for that computation to hide the loading time, k must approach the bandwidth ratio above, so we split each expert into smaller ones and route to more of them, which raises k and shrinks every swap at fixed active parameters and FLOPs (Dai et al., 2024; Krajewski et al., 2024). Other methods, such as cache-conditional experts (Skliar et al., 2025), can be tuned to

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.

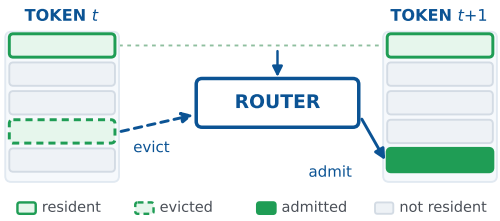


Figure 1: Temporal routing across one token boundary. Only the k resident experts ($k = 2$ of a five-expert pool here) (green) can serve, and at most one changes per layer per token, and the worst-scoring resident is evicted (dashed) for the best-scoring non-resident (solid), while the rest carry over. Fast memory therefore holds k experts rather than the pool, and the incoming expert streams in for free behind the compute of those already there.

average one expert load per layer but still require multiple loads for roughly one fifth of tokens. These additional loads delay decoding, while further reducing swaps sharply lowers accuracy (Fig. 8).

Our models train with our Temporal MoE constraint, at 18 activated of 192 experts, with the loss unchanged. A router trained this way relies less on token identity, spreading each expert's use across the token stream, and our findings suggest its experts are more general. Across four orders of magnitude of pretraining compute, the constrained model preserves most of the MoE-over-dense gain with $\leq 10\%$ of its experts in memory, and serving measurements confirm the memory savings on a workstation GPU, a phone, and a laptop.

The Temporal MoE constraint does not require pretraining from scratch. Imposed at decode time on released large instruct MoEs, with the prompt processed freely and the rule applied to generated tokens. This costs the strongest models a few points on generative benchmarks at the tightest setting, and less the more experts a model has. Furthermore, a lightweight two-GPU-hour distillation (training the constrained model to match its free-routing distribution on sampled responses) recovers about half of the lost accuracy for both Gemma4-26B and Qwen3.5-35B.

We make four contributions:

1. We introduce Temporal MoE, where each MoE layer replaces at most one of its active experts between consecutive tokens, allowing only active experts to stay in fast memory for serving. Across matched training compute budgets from 10^{16} to 10^{19} FLOPs, it retains most of the MoE advantage over dense models, with similar recovery in zero-shot accuracy.
2. We show how the constraint changes routing and expert weights, reducing reliance on token identity, spreading expert use more evenly, and improving quantization.
3. We establish the practical memory–latency tradeoff through serving measurements on a workstation GPU, phone, and laptop.
4. We demonstrate the remarkable robustness of six released large instruct MoEs to the constraint without adaptation, and show that lightweight distillation recovers most of the lost accuracy in Gemma4-26B and Qwen3.5-35B.

2 METHOD

Temporal MoE is one rule applied to a standard MoE layer, at pretraining time or at serving time.

Temporal Router. Each MoE layer with E routed experts keeps only R of them in fast memory (the resident experts), and we set $R = k$, the layer's active count, unless otherwise stated. Write S_t for the resident set at token t and $r_t \in \mathbb{R}^E$ for the router scores. The layer nominates the best non-resident expert, $n_t = \arg\max_{e \notin S_{t-1}} r_t[e]$, and swaps it in only if it beats the worst resident, which is evicted,

$$S_t = \begin{cases} \left(S_{t-1} \setminus \{\arg\min_{e \in S_{t-1}} r_t[e]\}\right) \cup \{n_t\}, & r_t[n_t] > \min_{e \in S_{t-1}} r_t[e], \\ S_{t-1}, & \text{otherwise.} \end{cases} \tag{1}$$

S_t replaces the router's top-k selection and nothing else about the layer changes. For the first token's set, S_0, we simply use its top-k. The language-modeling, load-balancing, and z-losses are unchanged. The swap itself is not differentiable, but the router still receives ordinary gradients through the scores it assigns to the experts, exactly like standard MoEs. The selection scan runs as a single-launch Triton kernel and costs about 1 ms per layer call in training, under 1% of step time at our scales.

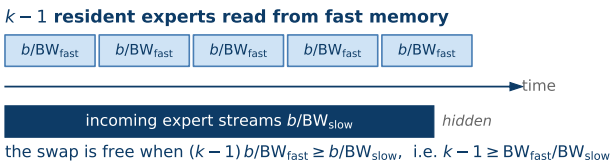


Figure 2: One decode step of one MoE layer under our temporal router. Decode is memory-bound, so reading the $k-1$ resident experts takes long enough to hide the incoming expert's stream beneath it. The expert size b cancels, so the routing width k alone decides whether the swap is free.

The bandwidth condition. Fig. 2 shows the timeline of one decode step. Decode at batch size 1 is memory-bound, and we analyze the cost profile through that lens. Serving the $k-1$ already-resident experts of size b reads $(k-1)\,b$ bytes from fast memory, while the incoming expert streams its b bytes from slower storage, and the swap is free when $b/\mathrm{BW}_{\mathrm{slow}} \le (k-1)\,b/\mathrm{BW}_{\mathrm{fast}}$, which simplifies to $k-1 \ge \mathrm{BW}_{\mathrm{fast}}/\mathrm{BW}_{\mathrm{slow}}$. Thus, for a given bandwidth ratio, the condition depends on the number of active experts k. We can increase k, while keeping the model the same size by making each expert smaller and choosing proportionally more of them. This quality cost of higher granularity's scaling laws is measured by Krajewski et al. (2024); Clark et al. (2022); Ludziejewski et al. (2025). Our fine-grained models route $k = 18$ of 192 experts plus an always-active shared expert, about $6\times$ the size of a routed one, which puts $k - 1 + 6 = 23$ within the 15-to-100 bandwidth ratios common between memory tiers. Section 5 profiles two real devices and tests the condition end-to-end. All serving decode measurements use batch size 1 because each sequence requires its own resident set, so resident expert memory can grow up to \$B\$-fold for a batch of \$B\$ sequences.

3 Pretraining under the constraint

Setup. We build on FLAME-MoE (Kang et al., 2025), an open MoE pretraining suite whose default layer routes 6 of 64 experts alongside one always-active shared expert, and train every model at two granularities, the FLAME default and the fine-grained split (18 of 192). At 10^{16} and 10^{17} FLOPs we compare model sizes at fixed compute budgets, spanning 0.77M to 15.1M active non-embedding parameters with a 16k vocabulary. At 10^{18} we sweep three sizes around an exact replication of FLAME-MoE-38M-100M (4.45B tokens, 50k vocabulary), three seeds at the best-performing size and one at each neighboring size. At 10^{19} we train the compute-optimal model size. Each configuration has a dense baseline matched to the MoE's active parameters and FLOPs. We use the same hyperparameters across model types and measure quality in tokenizer-invariant test-set bits per byte (BPB). Seed noise is only about 0.003 BPB. Lastly, recovery is $(\text{dense} - \text{temporal})/(\text{dense} - \text{MoE})$, the share of the MoE-over-dense gain that Temporal MoE keeps.

Quality at matched training compute. At every model size in the three budgets with full sweeps (Fig. 3), Temporal MoE performs between the dense and standard MoE baselines and keeps 72 to 82% of the MoE-over-dense gain at the compute-optimal sizes, 82% coarse in both smaller budgets and 72–77% fine, with the isoFLOP parabola and its minimum matching the baselines at every budget. At 10^{18} FLOPs the coarse temporal model reaches parity with the strongest full MoE (seed means 1.312 vs. 1.317 BPB) and the fine temporal model recovers 78% of the gain against it, in line with the smaller budgets, with seed spread under 0.004 BPB throughout (App. A). At 10^{19} the temporal models keep 78–81% of the gain against the strongest full MoE and 92% against the equally fine one.

Downstream accuracy. Zero-shot accuracy at 10^{19} (per-task table in App. A) shows similar recovery to the pretraining loss. The full MoE lifts the dense six-task average from 0.392 to 0.424 (0.059–0.091 above chance), and the temporal models keep 83% (coarse) and 76% (fine) of that gain.

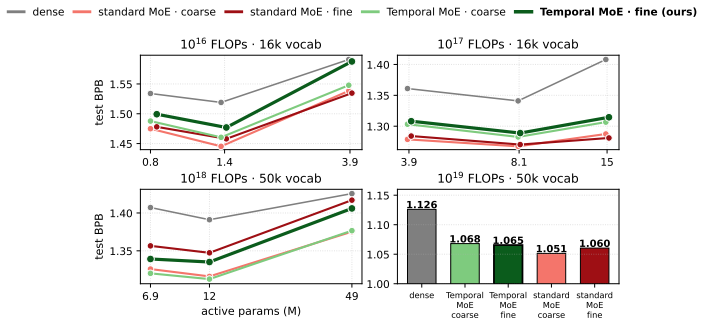


Figure 3: Test-set BPB at fixed compute, one panel per budget (lower is better). Gray dense, red standard MoE, green Temporal MoE, dark shades the fine granularity (18 of 192). Vocabularies differ between budgets, so compare within a panel, and the 10^{18} midpoints are three-seed means (App. A). Temporal MoE sits at or inside the dense-to-MoE band at every size and budget, keeping 72–82% of the gain at compute-optimal sizes with only its active experts resident.

In fact, the fine Temporal MoE is 97% of its fine standard MoE counterpart. The dense-to-MoE gap spans several standard errors while the MoE-to-temporal gaps sit within one, so the improvement over dense is clear, but the difference between standard and Temporal MoE is not statistically resolved.

Quality degrades smoothly with resident memory. Raising the resident budget R above the active k interpolates from the deployed setting to the full MoE at identical FLOPs, and because R is the memory held, Fig. 9 is also the serving memory-quality frontier. BPB falls monotonically, but the whole 10.7-fold range costs 0.023 BPB and most of it arrives in the last doubling toward $R = E$, so the frontier is flat where we deploy.

Auxiliary losses do not replace the bound. The constraint keeps 182–186 of 192 experts effective at every R, while the losses we tried in its place, each rewarding cache-friendliness, all lost quality and reduced that count, including CoSMoEs' block-selection loss. Even at strengths that raise BPB by up to 24%, CoSMoEs' loss only brings swaps down to 7.3 per token-layer, against the 1.0 we serve at, and collapses the effective experts to 36 of 192 (App. B).

4 What the router learns instead

Section 3 measured the quality lost under the constraint. Here we examine how it changes expert selection. Every measurement runs on a fixed evaluation batch per model, probes are fit per expert and scored on held-out documents (chance 0.5), and protocols, null controls, and tables are in App. C.

The router reuses experts even without the constraint. Fig. 4 (left) shows which experts are active at each token over a long sequence, at the deepest MoE layer of an 8.1M-active model. The standard MoE has scattered expert choices. Temporal MoE forms long streaks of repeated selections, even when evaluated without the residency mask. The trained router therefore learns to reuse experts beyond what the constraint requires.

The resident set anticipates demand, more so with depth. Hit rate is the share of a token's unconstrained top-k already resident before any swap, with $k/E = 0.094$ expected for a random resident set. The temporal router reaches 0.32 vs. 0.17 for the matched full MoE (Fig. 4, right). Hit rate also rises with depth in both regimes, roughly doubling from the first MoE layer to the last.

Token identity fades from the routing signal. For each expert, we train two probes predicting its use: one from the current token's embedding alone and one from the surrounding tokens' mean (CBOW-like, current token excluded). Across 34 trained models, token-based probe scores completely separate the two model types, with a gap of 0.184 between them, while context-based probes score similarly. Median token AUC falls from 0.884 in the unconstrained models to 0.585 in the temporal models while context AUC moves only from 0.640–0.697 (App. C), so the constraint mostly removes

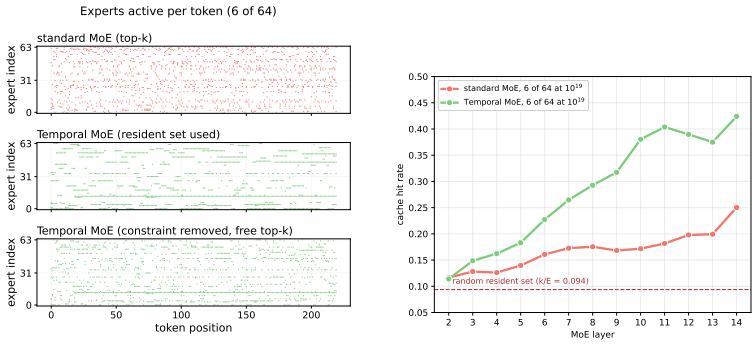


Figure 4: **Left,** experts active per token in the 8.1M-active model's deepest MoE layer, where the standard MoE scatters and both Temporal MoE traces streak. **Right,** cache hit rate by MoE layer against the $k/E = 0.094$ random floor (higher is better) for the matched 6-of-64 pair at 10^{19}. The preference for stable experts is inductively trained in rather than forced step by step, and the resident set anticipates demand about twice as well in the deepest layers as in the shallowest.

the token signal rather than adding a context signal. Figure 5 plots the difference directly against depth, where both grow more contextual, but temporal routing is contextual from the shallowest MoE layer on.

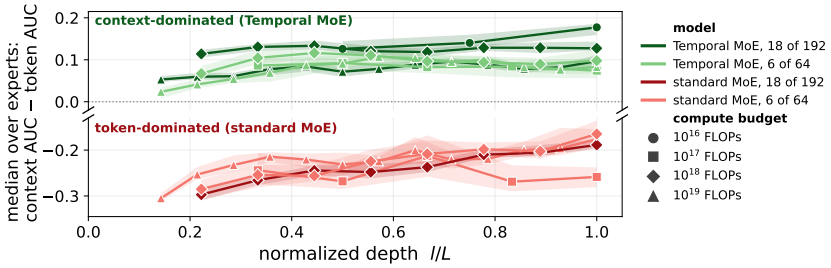


Figure 5: Median context-minus-token AUC per MoE layer against normalized depth, 95% bootstrap bands, broken y-axis; the key gives colour (family and granularity) and marker (compute budget). Temporal expert selection is more contextual than token-determined, the unconstrained MoE's is the reverse, and both grow more contextual with depth.

Experts spread across the stream. The participation ratio (the normalized inverse Simpson index) measures how evenly an expert's routing mass is distributed across tokens. The median rises from 30% in the full MoE to 66% in Temporal MoE at fine granularity, and from 34% to 52% at coarse granularity.

Experts become more replaceable and more distinct. Replacing one of six active experts (three of eighteen) with a random alternative costs Temporal MoE 18% less than standard MoE across 10^{17}-10^{19} FLOPs (App. C). While weight geometries remain indistinguishable, their outputs on identical inputs are 29–53% less alike at 10^{18} and 10^{19} (mean output cosine 0.028 vs. 0.041 at 10^{18} coarse, where random experts read 0.000), with peak layer similarity reaching only 50–66% of the baseline. Temporal MoE thus trains broader experts that substitute better despite computing less similar functions.

Temporal weights quantize better. Temporal MoE expert weights have fewer outliers (as measured by kurtosis) in every paired comparison. Temporal MoE also loses less quality under 4 and 3 bit quantization in every paired comparison (App. C). In the memory constrained use cases that Temporal MoE is designed for, it naturally facilitates better quantization, saving further memory.

5 SERVING

Setup and devices. We implement Temporal MoE in a llama.cpp CUDA fork and benchmark batch-1 prefill and decode on random-weight Qwen3-MoE-shaped models of matched total size ($\sim$11.3B, Q4_K_M) at both granularities, since only latency and memory are measured. On an RTX A6000, $R = k$ expert slots live in VRAM, the full pool in CPU RAM, and at most one expert per layer per token streams across PCIe (840 KiB per fine swap, 2.5 MiB per coarse), with prefill run expert-major so each needed expert streams once for all of its tokens. On a Pixel 10a and an i7-11370H laptop, the same discipline runs CPU-only, with R slots in RAM and experts read from UFS flash and NVMe storage, respectively.

Table 1: Decode throughput at 1024-token context (tok/s, RTX A6000, batch 1). Deploy holds only the $R = k$ active experts in VRAM, 1502 vs. 7672 MiB, and streams the rest from a CPU pool. Vanilla offload is the same memory without the trained constraint, free top-k routing with every miss fetched (App. D). At fine grain the deploy row (bold) keeps 0.83× of the all-resident ceiling in 5.1× less memory, where free routing at that memory keeps 0.22×.

setup	fine 18-of-192	vs ceiling	coarse 6-of-64	vs ceiling
ceiling: all experts in VRAM	200	1.00×	252	1.00×
our kernel, no swaps	176	0.88×	217	0.86×
deploy: active-only in VRAM	**165**	**0.83**×	128	0.51×
vanilla offload: active-only, free routing	43	0.22×	42	0.17×

Decode across devices. Table 1 quantifies the bandwidth condition behind Temporal MoE on an RTX A6000. At fine grain the deploy row decodes at 0.83× the all-resident speed with 5.1× less VRAM. Although for weights alone, we save 5.7×, KV cache and workspace buffers lower it to 5.1. Most of the slowdown with fine experts comes from the kernel changes. Without transfers, our kernel retains 0.88× the all-resident speed, falling to 0.83× when transfers are enabled. However, larger experts incur substantially more loading overhead, reducing speed from 0.86× to 0.51×. As a baseline, free routing at the same memory budget misses most of its experts on every token and runs 3–4× slower. Furthermore, on a Pixel 10a the temporal constraint is what lets the model run at all. All-resident needs 5.5 GiB and the phone has 3.6 GB usable, so the stock engine cannot load it. With only each layer's 18 active experts of 192 resident in RAM and the rest streamed from flash at a prescribed one swap per layer per token, the model decodes at 17.8 tok/s at 1k context and 12.2 at 4k, 70 and 83% of a ceiling measured on the largest expert pool that does fit (App. D). On an Intel i7-11370H laptop, the model decodes at 18.7 tok/s at 1k context and 9.9 tok/s at 4k, 66% and 75% of the baseline MoE's speed, and its decode memory falls 5.6×, from 5968 to 1066 MiB (App. D).

Prefill and long context. On the workstation, Fig. 6 extends the measurement across context lengths. Prefilling 16k tokens takes 1.6× the all-resident time, decode holds near 0.80× throughout, and peak VRAM stays under 3.6 GiB vs. 8.0 to 9.3 GiB all-resident, a gap the growing KV cache compresses from 3.7× at 1k tokens to 2.6× at 16k. At larger batch each sequence carries its own resident set, so the advantage narrows there as well. Micro-batch choice, the prefill decomposition, and the vanilla-offload floor are in App. D.

6 RELEASED INSTRUCT MODELS UNDER RESIDENCY

Setup. We impose the decode-time protocol of §2 on six released instruct MoEs, by total size OLMoE-1B-7B-Instruct (64 experts, $k = 8$), LFM2.5-8B-A1B (32, $k = 4$), GPT-OSS-20B (32, $k = 4$), Gemma4-26B-IT (128, $k = 8$), Qwen3.5-35B-A3B (256, $k = 8$), and GPT-OSS-120B (128, $k = 4$), each under free routing, $R = k$, and $R = 12.5\%$ of the pool, the tightest fraction every model shares, in both thinking modes where a model has them. We benchmark GSM8K, IFEval, HumanEval, MMLU, MBPP, and WritingBench each for a single run which has standard errors of 1–4 points, and the full protocol is in App. E.

Accuracy losses vary and shrink with more experts resident. Released models vary considerably in how well they tolerate the residency constraint without further training (Fig. 7, top). With thinking

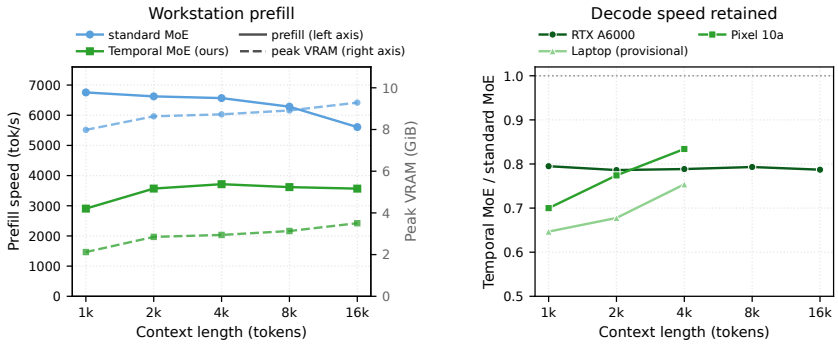


Figure 6: Serving across context lengths. Left, workstation prefill speed and peak VRAM. Right, decode speed relative to each device's standard MoE baseline, using matched context lengths. The phone baseline is fixed at 80 of the 192 experts per layer, the largest expert pool that fits in memory. Across devices, our technique consistently maintains 70–80% speed while significantly cutting RAM use. (App. D).

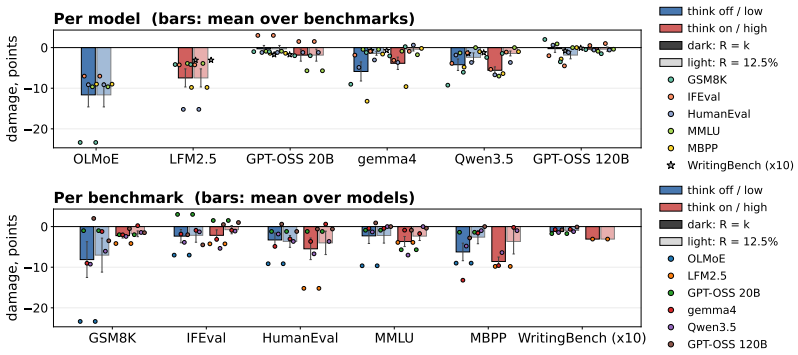


Figure 7: Damage under decode-time residency (constrained minus free, points), WritingBench critic deltas $\times 10$, panels labeled in place. Dots are the individual values behind each mean, whiskers its standard error, single runs with binomial standard errors of 1–4 points. Every model survives the constraint without training, some losing under half a point, and keeping more experts resident shrinks the loss further.

disabled or effort set to low, keeping only $R = k$ experts resident reduces mean accuracy across the five benchmarks by 11.6 percentage points for OLMoE, 7.5 for LFM2.5, 5.9 for Gemma4, and 4.2 for Qwen3.5. Both GPT-OSS models lose just 0.3 points, despite keeping only as many experts resident as they activate per token. These averages can hide differences between tasks. Gemma4 loses 13.2 points on MBPP but averages a 4.0-point loss across the other four benchmarks. WritingBench scores fall by only 0.07 to 0.31 points on the 1-to-10 critic scale at $R = k$, so writing tasks appear less sensitive to the constraint. Increasing residency to 12.5% of the expert pool recovers much of the lost accuracy for Gemma4 and Qwen3.5, bringing their mean scores to 1.4 and 2.4 points below their free-routing baselines, respectively. The same resident count also represents different constraints across models. At $R = 8$, Qwen3.5 retains only 3.1% of its experts, compared with Gemma4's 6.25%, yet its mean accuracy loss is smaller.

Thinking changes residency losses modestly. Enabling thinking changes the mean accuracy loss at $R = k$ only modestly (Fig. 7, top). Against free routing in the same thinking mode, Gemma4 loses 3.9 percentage points with thinking enabled versus 5.9 disabled. Qwen3.5 loses 5.6 versus 4.2 points. Raising reasoning effort from low to high changes the loss from 0.3 to 0.4 points for GPT-OSS-120B and from 0.3 to 1.9 for GPT-OSS-20B. Gemma4 loses 13.2 points on MBPP with thinking disabled. With thinking enabled, Qwen3.5 loses 2.4 to 7.0 points per benchmark. Increasing its resident fraction to 12.5% reduces its mean loss to 1.4 points.

Table 2: On-policy self-distillation at R=8, absolute scores (GSM8K n=1319, IFEval 541, MMLU 228, HumanEval 164, MBPP 500). mean$_5$ averages the five columns, and the comparisons at the same resident fraction are in Table 10. Two GPU-hours of distillation bring the constrained models within 3.0 and 1.8 points of the original model.

		GSM8K	IFEval	MMLU	HumanEval	MBPP	mean$_5$
gemma4	released, free	87.8	88.7	93.0	99.4	91.2	92.0
	released, R=8	78.8	86.9	92.5	94.5	78.0	86.1
	adapted, R=8	82.6	86.3	94.3	97.0	85.0	89.0
Qwen3.5	released, free	85.9	86.5	93.4	92.7	80.0	87.7
	released, R=8	76.6	82.6	92.1	90.9	75.2	83.5
	adapted, R=8	83.5	82.6	91.7	92.7	79.0	85.9

With thinking enabled, Gemma4 generates responses 16% to 21% longer at $R = 8$ than under free routing across GSM8K, IFEval, and HumanEval. Qwen3.5 shows a similar increase of 12% to 24% on these tasks (Fig. 19). Appendix F gives the full length analysis.

7 Adapting released models

We adapt Gemma4 and Qwen3.5 through on-policy self-distillation, using each model's original checkpoint as its teacher. The student generates responses with $R = 8$ experts resident, while the frozen teacher evaluates the same token sequences with free routing. We train the student to match the teacher's next-token distribution by minimizing reverse KL divergence. Two GPU-hours of adaptation improve accuracy under the residency constraint while changing free-routing accuracy by at most half a percentage point. Training uses prompts from WildChat (Zhao et al., 2024), StackMathQA (Zhang, 2024), and OASST2 (Köpf et al., 2023), screened for 8-gram overlap with every test set. The pool covers general instruction, coding, mathematics, and few-shot tasks. Appendix E gives the training details and Table 9 lists the prompt counts and composition.

Adaptation recovers about half the mean accuracy lost at $R = 8$ for both models (Table 2). Gemma4 gains 2.9 percentage points and Qwen3.5 gains 2.4. On GSM8K, they recover 3.9 and 6.9 points, respectively. Gemma4's largest gain is on MBPP, its most affected benchmark, where it recovers 7.0 points. No benchmark declines beyond run-to-run noise. All comparisons use the same vLLM engine class to avoid differences caused by the inference implementation (App. E). At 12.5% residency, adaptation yields smaller gains because the original losses are smaller (Table 10). With thinking enabled, adaptation still improves coding performance. Gemma4 gains 8.4 points on MBPP and Qwen3.5 gains 6.7 on HumanEval, both statistically significant improvements. The GSM8K changes, however, are not statistically distinguishable from noise (Table 11). WritingBench scores change by at most 0.16 critic points. Adaptation also removes the increase in repeated three-token sequences seen in Qwen3.5 under the residency constraint.

Adaptation reduces the extra generation length caused by residency more consistently for Gemma4 than for Qwen3.5 (Fig. 19). With thinking enabled, Gemma4's constrained responses after adaptation are within 12% of the original free-routing lengths, while unconstrained responses remain close to that baseline. For Qwen3.5, adaptation narrows the difference between its constrained and free-routing lengths, but also lengthens its free-routing responses by 11% on IFEval and 26% on HumanEval.

Serving-cache baselines at matched memory. Figure 8 compares our method with cache-conditional experts (Skliar et al., 2025) and ReMoE's reuse finetuning (Zhu et al., 2026). Cache-conditional routing preserves accuracy when half the experts remain resident, but loses accuracy at our smaller memory budgets of 6.25% for Gemma4 and 3.1% for Qwen3.5. At the settings closest to our one-swap budget, its GSM8K scores are 80.7 for Gemma4 and 74.5 for Qwen3.5, compared with 82.6 and 83.5 after our adaptation. Reducing transfers further lowers its scores to 69.5 and 45.1, respectively. At these memory budgets, encouraging the router to reuse cached experts therefore imposes a tradeoff between accuracy and transfer rate.

Average transfer rates also hide variation between tokens and layers. At a cache-conditional setting averaging 0.94 expert transfers per token per layer, roughly one fifth of token-layer pairs require two

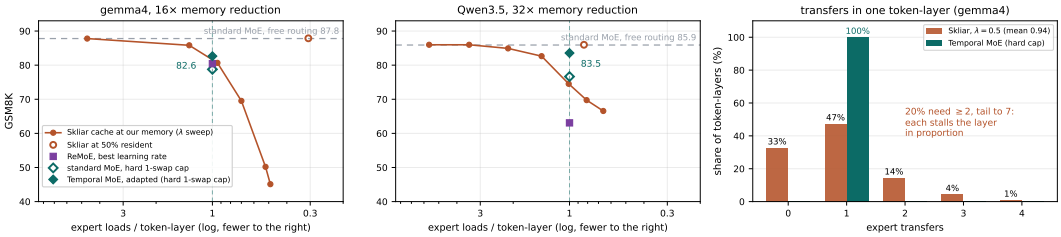


Figure 8: Serving-cache baselines. **Left and center,** GSM8K (n=1319) against expert loads per token-layer measured on the eval generations, every method at our resident memory (6.25% gemma4, 3.1% Qwen3.5) except the hollow circles, Skliar's cache at its own 50% setting, and the square, ReMoE at its best learning rate under the bound. **Right,** per-token-layer transfer distribution for Skliar's cache (mean 0.94) versus our constant 1-swap cap. Matching our accuracy requires exceeding this budget (>1 load/token-layer), with 20% of token-layers needing multiple transfers in Skliar."

or more transfers, with some requiring seven. These bursts can stall decoding even when the average fits a one-transfer budget. Our rule limits every token-layer pair to at most one transfer, giving a fixed upper bound on transfer demand. When the bandwidth condition holds, resident computation can hide that transfer entirely (Sections 2 and 5). A natural extension is to add their cache bias to our method, with λ controlling the penalty for transfers beyond the first.

8 Related work

Serving systems keep a pool of $R > k$ experts in fast memory and manage the misses, with prefetching caches (Eliseev & Mazur, 2023; Xue et al., 2024; Song et al., 2024), early gating that overlaps the fetch (Hwang et al., 2024), or a bias that steers a frozen router toward the cache (Skliar et al., 2025), surveyed in Liu et al. (2026). Locality there is a tendency and the pool still provisions for misses, a difference §7 prices at 16–32× expert-memory reductions. Temporal MoE trains under the footprint and swap budget, so §2's bandwidth condition holds by construction.

A newer line shapes the routing itself, with a compact semantic routing space (Zhou et al., 2025), losses that penalize switches or shrink the per-chunk expert union (Kayyam, 2026; Song et al., 2025; Huber et al., 2026), router-only reuse finetuning (Zhu et al., 2026; Raje et al., 2026), and a learned controller for when the expert set switches (Shen & Henderson, 2026). The objectives we ran cost quality and expert diversity for little reuse (App. B), and reuse finetuning fails under a hard bound, such as our single swap (§7).

A complementary axis serves capacity whose lookup key is known before compute runs, through token-keyed expert tables (Jie et al., 2025), per-layer embeddings in Gemma's edge models (Gemma Team, 2026), and N-gram-keyed memory (Cheng et al., 2026). Lookup capacity is lexical by construction and token-identity binding is what residency removes, so the two are complementary.

Path-constrained MoE ties expert choices across neighboring layers (Gu et al., 2026) and Mixture-of-Depths spends sparsity on skipping compute across the sequence (Raposo et al., 2024). Residency is the time-axis member of that family, aimed at the cost that binds at deployment.

9 Conclusions

Across four orders of magnitude of training compute, two tokenizer families, and two expert granularities, TemporalMoE keeps 72–82% of the MoE-over-dense accuracy improvement with only the active experts in fast memory. Quality holds because routing reorganizes around context rather than token identity, producing smoother expert weights that are cheaper to quantize. For our pretrained models, TemporalMoE cuts serving memory about 5× while retaining 66–83% of decode speed across a workstation GPU, a phone, and a laptop. Released large models can tolerate the constraint out of the box, while just two GPU-hours of distillation closes most of the remaining accuracy gap. In all, training expert reuse into the model advances the quality–memory frontier for local inference, delivering 7–10× memory reductions on released models while preserving both speed and accuracy, a trade-off unmatched by existing caches or reuse objectives.

AI USE STATEMENT

We used AI coding agents to iterate quickly on the experiment harnesses, the analysis scripts, and the figure producers behind this work. The authors reviewed the generated code and the results it produced, and the core temporal router was written by the authors by hand. AI was further used for small writing polishes on-top of what the authors already wrote.

ETHICS STATEMENT

This work aims to make strong open-weight models servable on local, private hardware, reducing dependence on hosted inference and its data-sharing requirements. All evaluated models and benchmarks are publicly released artifacts used under their licenses, no human subjects or private data are involved, and the serving techniques carry the same dual-use profile as the open-weight models they make more accessible.

REPRODUCIBILITY STATEMENT

Every figure and table regenerates from a committed producer script reading committed result files. Pretraining runs are specified by a shape and a FLOP budget through a single launcher with locked, recorded hyperparameters, and the published artifacts, including checkpoints, training logs, and router captures, are indexed by a manifest with sizes and checksums. The instruct-model grid stores per-item generation dumps keyed by document id, so scoring changes are re-analyses rather than regenerations, and the serving measurements ship with the engine patch, model-construction scripts, and benchmark drivers. The baseline reimplementations, the cache bias, the reuse objective, and the block-selection loss, follow the same rule, with each plotted point a named record in the committed result tables. Code, the artifact manifest, and the dumps will be released publicly upon publication.

REFERENCES

Xin Cheng et al. Conditional memory via scalable lookup: a new axis of sparsity for large language models. *arXiv:2601.07372*, 2026.

Aidan Clark, Diego de las Casas, Aurelia Guy, Arthur Mensch, Michela Paganini, Jordan Hoffmann, Bogdan Damoc, Blake Hechtman, Trevor Cai, Sebastian Borgeaud, George van den Driessche, Eliza Rutherford, Tom Hennigan, Matthew Johnson, Katie Millican, Albin Cassirer, Chris Jones, Elena Buchatskaya, David Budden, Laurent Sifre, Simon Osindero, Oriol Vinyals, Jack Rae, Erich Elsen, Koray Kavukcuoglu, and Karen Simonyan. Unified scaling laws for routed language models. In *International Conference on Machine Learning (ICML)*, 2022.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2024.

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

Gemma Team. Gemma 4 technical report. *arXiv:2607.02770*, 2026.

Zijin Gu et al. Path-constrained mixture-of-experts. *arXiv:2603.18297*, 2026.

Patrick Huber, Akshat Shrivastava, Ernie Chang, Chinnadhurai Sankar, Ahmed A Aly, and Adithya Sagar. CoSMoEs: compact sparse mixture of experts. In *Proceedings of the 4th Workshop on Advances in Language and Vision Research (ALVR)*, 2026.

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 *International Symposium on Computer Architecture (ISCA)*, 2024.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv:2401.04088*, 2024.

Shibo Jie, Yehui Tang, Kai Han, Yitong Li, Duyu Tang, Zhi-Hong Deng, and Yunhe Wang. Mixture of lookup experts. In *International Conference on Machine Learning (ICML)*, 2025.

Hao Kang, Zichun Yu, and Chenyan Xiong. FLAME-MoE: a transparent end-to-end research platform for mixture-of-experts language models. *arXiv:2505.20225*, 2025.

Ali Kayyam. Sticky routing: training MoE models for memory-efficient inference. *arXiv:2607.08780*, 2026.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. OpenAssistant conversations – democratizing large language model alignment. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Jakub Krajewski, Jan Ludziejewski, Kamil Adamczewski, Maciej Pióro, Michał Krutul, Szymon Antoniak, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Piotr Sankowski, Marek Cygan, and Sebastian Jaszczur. Scaling laws for fine-grained mixture of experts. In *International Conference on Machine Learning (ICML)*, 2024.

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

Jiacheng Liu, Peng Tang, Wenfeng Wang, Yuhang Ren, Xiaofeng Hou, Pheng-Ann Heng, Minyi Guo, and Chao Li. A survey on inference optimization techniques for mixture of experts models. *ACM Computing Surveys*, 2026.

Jan Ludziejewski, Maciej Pióro, Jakub Krajewski, Maciej Stefaniak, Michał Krutul, Jan Małaśnicki, Marek Cygan, Piotr Sankowski, Kamil Adamczewski, Piotr Miłoś, and Sebastian Jaszczur. Joint MoE scaling laws: mixture of experts can be memory efficient. In *International Conference on Machine Learning (ICML)*, 2025.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open mixture-of-experts language models. In *International Conference on Learning Representations (ICLR)*, 2025.

Qwen Team. Qwen3.5-35b-a3b, 2026. Model card, `https://huggingface.co/Qwen/Qwen3.5-35B-A3B`.

Arian Raje, Anupam Nayak, and Gauri Joshi. MELINOE: fine-tuning enables memory-efficient inference for mixture-of-experts models. *arXiv:2602.11192*, 2026.

David Raposo et al. Mixture-of-depths: dynamically allocating compute in transformer-based language models. *arXiv:2404.02258*, 2024.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017.

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

Andrii Skliar, Ties van Rozendaal, Romain Lepert, Todor Boinovski, Mart van Baalen, Markus Nagel, Paul N. Whatmough, and Babak Ehteshami Bejnordi. Mixture of cache-conditional experts for efficient mobile device inference. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=ul4W26KEKz.

Chenyang Song, Weilin Zhao, Xu Han, Chaojun Xiao, Yingfa Chen, Yuxuan Li, Zhiyuan Liu, and Maosong Sun. BlockFFN: towards end-side acceleration-friendly mixture-of-experts with chunk-level activation sparsity. In *Conference on Language Modeling (COLM)*, 2025.

Xiaoniu Song et al. ProMoE: fast MoE-based LLM serving using proactive caching. *arXiv:2410.22134*, 2024.

Leyang Xue et al. MoE-Infinity: efficient MoE inference on personal machines with sparsity-aware expert cache. *arXiv:2401.14361*, 2024.

Yifan Zhang. StackMathQA: A curated collection of 2 million mathematical questions and answers sourced from stack exchange, 2024. Dataset, https://github.com/yifanzhang-pro/StackMathQA.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. WildChat: 1m ChatGPT interaction logs in the wild. In *International Conference on Learning Representations (ICLR)*, 2024.

Jixian Zhou et al. Oracle-MoE: locality-preserving routing in the oracle space for memory-constrained large language model inference. In *ICML*, 2025.

Xiongwei Zhu et al. ReMoE: boosting expert reuse through router fine-tuning in memory-constrained MoE LLM inference. *arXiv:2605.27081*, 2026.

A Pretraining detail: seeds, granularity, and downstream tasks

Table 3 lists the runs behind the 10^{18} midpoint of Fig. 3, all on the same data split.

Table 3: Per-seed test BPB at the 10^{18} compute-optimal midpoint (lower is better). The dense floor is a single run.

	seed			
model	1	2	3	mean
temporal, 6 of 64	1.3128	1.3111	1.3128	1.3122
full MoE, 6 of 64	1.3158	1.3197	1.3169	1.3175
temporal, 18 of 192	1.3354	1.3339	1.3323	1.3339
full MoE, 18 of 192	1.3461	1.3489	1.3483	1.3478
dense	1.3911			1.3911

A single-variable extension trains both paradigms at 30 of 320 experts on the same budget and split. The temporal model reaches 1.3491 vs. 1.3625 for the full MoE, so the temporal advantage at this granularity (0.013 BPB) matches the replicated 18-of-192 gap (0.014) and sits above the observed seed ranges of 0.003–0.004, while the 6-of-64 gap (0.005) sits near them. Both 320-expert models still beat the dense floor. These two are single seeds, quoted because they land where the replicated gap predicts, and fine-graining costs absolute quality at this budget even where the temporal-over-MoE gap widens.

Table 4 lists the per-task accuracies behind §3's downstream averages.

B Negative results

Objectives that reward cache-friendliness. Three trained objectives tried to buy locality with a loss instead of the hard constraint, and all three lost quality. An explicit loss pulling router logits

Table 4: Zero-shot accuracy at 10^{19} FLOPs (lm-evaluation-harness, acc_norm except WinoGrande, which reports acc). Avg−chance subtracts each task's chance floor (0.5 for WinoGrande and PIQA, 0.25 elsewhere) before averaging. Bold marks the best per column. Per-task standard errors are 0.004–0.021, so single-task bold margins of a point or two sit within noise and the aggregate columns carry the comparison.

model	OBQA	WinoG	PIQA	ARC-e	ARC-c	HSwag	Avg	Avg−chance
dense	.276	.501	.621	.437	.222	.296	.392	.059
full MoE, 6 of 64	.296	**.509**	.658	**.495**	.245	**.341**	**.424**	**.091**
full MoE, 18 of 192	**.308**	**.509**	.657	.445	.247	.339	.417	.084
temporal, 6 of 64	.292	**.509**	**.659**	.471	.246	.336	.419	.085
temporal, 18 of 192	.290	.486	.651	.484	**.253**	.335	.417	.083

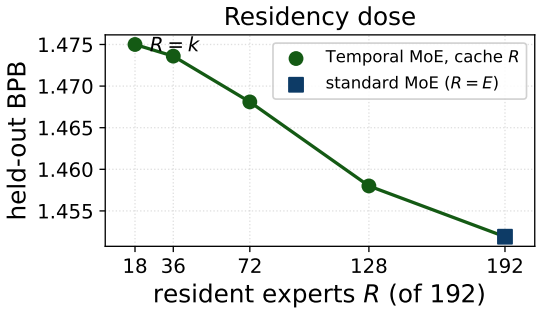


Figure 9: Held-out BPB against resident budget R at identical FLOPs (192 experts, $k = 18$, 10^{16} FLOPs, lower is better). Because R is the resident memory this is the serving memory-quality frontier, and it is flat where we deploy, with most of the 0.023 BPB range arriving in the last doubling toward $R = E$.

toward the current resident set hurts monotonically at every weight scanned at 10^{16} FLOPs, at both granularities. An auxiliary-loss-free selection bias with demand momentum raises next-token overlap by 7 points at its best setting while inducing a de facto pinned expert, resident 85% of the time, with no quality gain. An anticipatory loss on discounted future demand raises alignment by 26 points by teaching the router to concentrate demand, and streamed expert diversity collapses from a union of 158 experts to 50 of 192 while BPB worsens. The pattern across all three is that hit rate bought by concentrating usage costs quality, while the trained-in constraint raises hit rate and spreads usage at once.

The published member of this family. CoSMoEs (Huber et al., 2026) penalizes block-level expert diversity so that a model trained from scratch switches experts less often. We trained its loss (their Eq. 4–7, unit-tested) on a vanilla MoE in the 10^{16} venue at both granularities, with weights from 0.01–100 because the paper states none. Table 5 gives the sweep. At the weight whose quality tax matches ours ($+0.8$ to $+0.9\%$ BPB) the demanded loads barely move, 12.9 per token-layer against the unpenalized 14.1 at fine grain. Heavier weights shed loads only by raising BPB to $+24\%$ and collapsing the effective pool from 191 to 36 of 192 experts, and the sweep never drops below 7.3 loads per token-layer. The demanded set never becomes bounded, so no weight can serve at $R{=}k$; the hard constraint provides that at a $+1\%$ tax with the pool intact. Loads are router demand probed on a fixed batch, the experts that must load if only the previous token's set stays resident, one seed per cell.

The pool is not starved. Backing §4's usage claims, a temporal model touches 84–100% of its expert pool over a 2048-token sequence across every shipped configuration, effective expert counts sit within 7% of the pool size, and no expert is resident more than 40% of the time, no more than 25% at 10^{19}.

Table 5: CoSMoEs' block-selection loss swept in the 10^{16} venue, each cell coarse (6 of 64) / fine (18 of 192) grain. Loads are router demand on a fixed batch; the temporal models serve at a hard 1.0 regardless of demand.

cell	loads / token-layer	effective experts	BPB tax
vanilla MoE	4.79 / 14.13	63.7 / 190.9	–
temporal (ours), raw demand	3.73 / 11.39	61.7 / 186.0	+1.0% / +1.2%
temporal (ours), served	1.00 / 1.00	61.7 / 186.0	+1.0% / +1.2%
BlES λ=0.01	4.42 / 12.85	59.4 / 157.8	+0.9% / +0.8%
BlES λ=0.03	3.70 / 11.93	43.1 / 120.0	+3.6% / +3.2%
BlES λ=0.1	3.14 / 10.19	20.5 / 68.4	+6.2% / +5.1%
BlES λ=1	3.28 / 10.44	19.8 / 64.6	+10.5% / +9.2%
BlES λ=10	2.48 / 8.58	12.5 / 43.4	+22.6% / +20.2%
BlES λ=100	1.91 / 7.33	10.2 / 36.1	+32.7% / +23.7%

The organization is trained into the weights. Each regime performs best as trained. Unmasking a trained temporal model, making every expert resident, costs +0.099 BPB at 10^{16}, +0.124 at 10^{17}, and +0.201 to +0.206 at 10^{19}. Imposing residency on a trained full MoE costs +0.240, +0.610, and +0.431 at the same budgets, and +0.595 for the fine full MoE at 10^{19}, two to five times more than unmasking. Only the unmasking direction rises with budget, so we claim no trend for imposition.

Early routing fails when trained in. Two overlap-widening architecture variants were screened at 10^{17} FLOPs behind flags whose off-state matched the unmodified trainer to within the run-to-run nondeterminism floor. Routing on the pre-attention state passed the screen (+0.004 BPB temporal, +0.000 full MoE, five MoE layers), and a parallel attention-and-MoE restructure failed it outright. Promoted to 10^{18} with eight MoE layers, the early-routing variant failed on both legs, +0.044 BPB on the temporal model and +0.150 on the full MoE, which lands above the dense floor. The damage is worse on the free-routing leg, so it is not temporal-specific, and it grows from five to eight MoE layers. The method lesson is that screens for per-layer insults must run at deployment depth. One seed per configuration.

C Probe protocols and structural measurements

Construction. All routing statistics derive from a fixed evaluation batch of 64 sequences of 2048 tokens per model. For each (layer, expert) pair, a ridge probe predicts whether the expert is in the token's served set, the top-k for the standard MoE and the resident set the constrained router uses for Temporal MoE, from one of two features: the current token's input embedding alone, or the mean embedding of the tokens within a window of k neighbors with the current token excluded, so context features cannot encode the token itself. Probes are fit and scored on disjoint documents, and chance is 0.500. Refitting on labels destroyed by iid permutation reads 0.500 ± 0.003 at worst over 1,162 fits, the floor every reported AUC is measured against.

Causal and ceiling checks. Two checks say §4's separation is routing rather than probe behavior. First, swapping the current token for a frequency-matched alternative with the context held fixed, against the reverse swap, moves temporal routing 1.58$\times$ as much through context as through the token (range 1.25 to 2.18), while the full MoE moves 0.69$\times$ as much (0.28–0.79), with 29 paired measurements per regime and non-overlapping ranges. Second, a nonparametric ceiling computes the best AUC achievable from token identity alone, and the linear token probe recovers 99–102% of it in both regimes, so the collapse of the token probe under the constraint reflects absent signal rather than a weak probe.

Substitution tolerance. On 512 cached test sequences per checkpoint (1.05M tokens), a hook after routing replaces active experts at every MoE layer and records the per-token cross-entropy change against the unperturbed pass, with the full MoE run through the same masked router with every expert resident, so both regimes share one code path, and the substitute taking the weight the router would assign it. At 10^{18}, replacing one of six active experts with a random other expert costs the coarse full MoE +0.044 BPB and its temporal counterpart +0.036, and three of eighteen costs the fine pair

+0.036 against +0.029, with the paired document bootstrap clear of zero for every seed (pair deltas -0.007 to -0.009, intervals within ± 0.0007). The same ordering holds at 10^{17} and 10^{19}, at a quarter of the full-MoE cost and a tenth or less respectively, and at 10^{19} the two regimes are equally tolerant of losing an expert while the full MoE stays far more sensitive to a wrong expert at real weight. Two readings keep the claim honest. A random substitute at its own weight costs almost what dropping the expert costs, so the headline measures tolerance to losing an active expert. Separating the two, dropping an expert costs the temporal model 17–20% less than the full MoE, and adding a random expert's output at the dropped expert's full weight costs it a quarter (coarse) and a fifth (fine) of what the full MoE pays, with the next-best expert at full weight costing nothing at coarse grain and helping slightly at fine. And the advantage belongs to the middle and late layers (Fig. 10). At the second MoE layer, where both regimes are most fragile, the temporal model is equal or slightly worse. The measurement says temporal experts are more replaceable, not that this is why quality survives the constraint. Nor is it because the experts converge. On one cached test micro-batch, with every routed expert evaluated on the same 2,048 token positions, the mean output cosine over all expert pairs is lower for the temporal models at 10^{18} and 10^{19} (0.028 against 0.041 coarse and 0.024 vs. 0.035 fine at 10^{18}), the peak layer sits at about half (0.045 vs. 0.069, 0.044 vs. 0.087), and both regimes lie within 0.02 of the random-init floor at 10^{17}. The gate spreads its mass nearly evenly over the selected experts, and the per-layer size of a swap perturbation does not predict the per-layer substitution cost (Spearman 0.07), so the tolerance is a property of the network downstream of the swap.

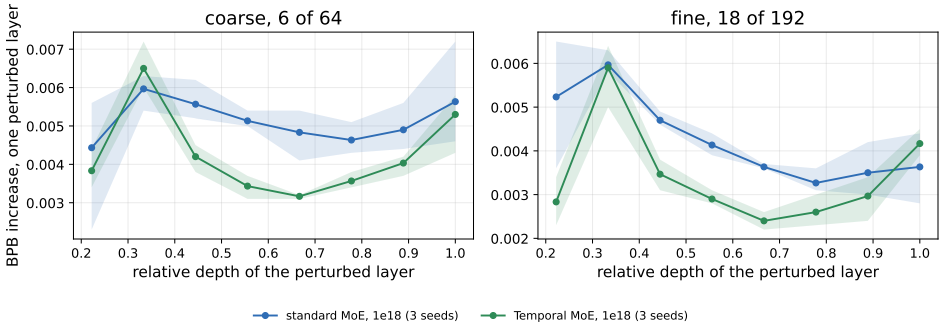


Figure 10: Substitution tolerance by depth at 10^{18}, the BPB increase from replacing one of six (three of eighteen) active experts in a single layer with a random other expert at its own router weight, bands spanning three seeds. The temporal advantage sits in the middle and late layers and closes at the second MoE layer, where both regimes are most fragile.

Structure across configurations. Table 6 gives the per-configuration statistics behind the usage claims. Participation ratio is the normalized inverse Simpson concentration of an expert's routing mass, generalists are experts above 0.5, and router flatness is per-token entropy normalized by $\ln E$. Independent seed replicates agree to about 0.01 in participation ratio and two points of generalist fraction, and the regime gaps are ten to forty times larger. Weight identity, the cosine distance of each expert's flattened weights to the expert centroid and the median pairwise cosine, is nearly unchanged between regimes.

Depth and the per-model view. Fig. 11 gives the per-model scatter behind §4's separation claim, whose per-layer difference Fig. 5 plots in the main text. Per-model slopes of context-minus-token AUC against layer index are +0.014 per layer for the unconstrained models, with bootstrap intervals excluding zero in four of five, against +0.002 for the temporal models, excluding zero in three of six, and the regime gap narrows from roughly 0.39 at the shallowest MoE layer to 0.27 at the deepest.

Weight statistics and quantization. Behind §4's summary, excess kurtosis of the routed-expert weight matrices, 0 for a Gaussian, is lower for the temporal model in every matched pair, with medians of 0.15 vs. 0.52 for the coarse pair and 0.32 against 0.82 for the fine pair at 10^{18}, and 0.08 vs. 0.13 and 0.14 vs. 0.22 at 10^{19}. The gap widens in the tail, where the fine 10^{18} pair's 99th percentile falls from 3.15–0.80. Fewer outliers make quantization cheaper. Under per-group round-to-nearest on routed experts only (Fig. 12), 8-bit is lossless in both regimes, and at 4 and 3 bits the temporal

Table 6: Routing structure and weight identity across configurations: median participation ratio, generalist fraction, normalized router entropy, mean cosine distance to the expert-weight centroid, and median pairwise cosine. The sigmoid control is the full MoE with a sigmoid router in place of the softmax.

model	part. ratio	generalist %	entropy	centroid dist	pairwise cos
full MoE, 18 of 192	0.30	1.7	0.88	0.90	0.005
sigmoid control (2 seeds)	0.27–0.28	5	0.87	0.89–0.90	0.007
temporal, 18 of 192 (2 seeds)	0.65–0.66	65–67	0.95	0.87–0.88	0.010–0.012
full MoE, 6 of 64	0.34	13	0.86	0.84	0.009
temporal, 6 of 64	0.52	54	0.93	0.84	0.012
full MoE, 6 of 64, 10^{19}	0.27	24	0.85	0.87	0.002
temporal, 6 of 64, 10^{19}	0.60	68	0.95	0.86	0.004
full MoE, 18 of 192, 10^{19}	0.18	10	0.84	0.91	0.003
temporal, 18 of 192, 10^{19}	0.51	53	0.95	0.91	0.003

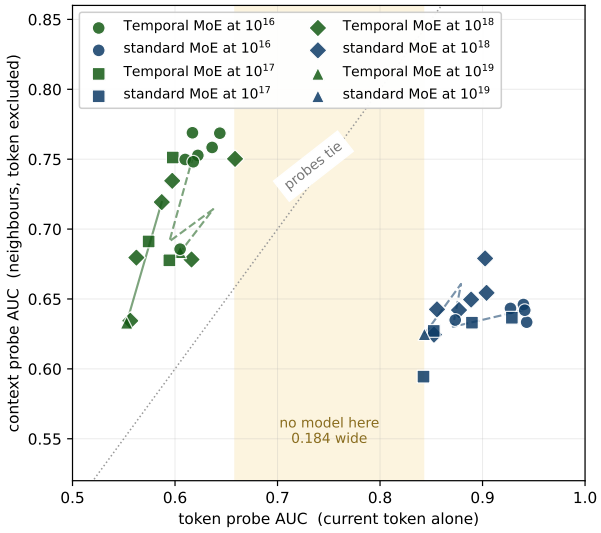


Figure 11: One point per trained model, 34 models across four budgets and both granularities. Horizontal, how well the current token alone predicts expert firing (median per-expert AUC, held out on unseen documents, chance 0.5). Vertical, the same for the surrounding context with the token excluded. The shaded band on the token axis is empty, 0.184 wide, and separates every temporal model from every unconstrained one, while the context axis separates nothing.

model degrades less than its matched full MoE in all four pairs, at both granularities and both scales measured. The two memory levers, residency and precision, compose rather than trade off, which matters because they are the obvious pair to reach for together.

D SERVING MEASUREMENT DETAILS

Workstation protocol. All RTX A6000 numbers use the llama.cpp CUDA fork of §5 at Q4_K_M, four trials with the first discarded, decode over a 100-token generation after the prefill, and a perplexity gate that verifies the streaming path bit-identical to the all-resident engine. The workstation's VRAM-to-PCIe bandwidth ratio is about 24–31, more than the serving model's 17 reused resident experts can hide on their own, but the copy still overlaps with the rest of each decode step, so decode only drops from 176 to 165 tok/s (Table 1).

Micro-batch choice. llama.cpp processes the prompt in micro-batches, and Table 7 sweeps the size at 16k context. Figure 6 runs both tiers at 2048, within 1% of the all-resident time optimum and near the deploy VRAM floor, so the setting flatters neither side. Larger micro-batches cost the deploy

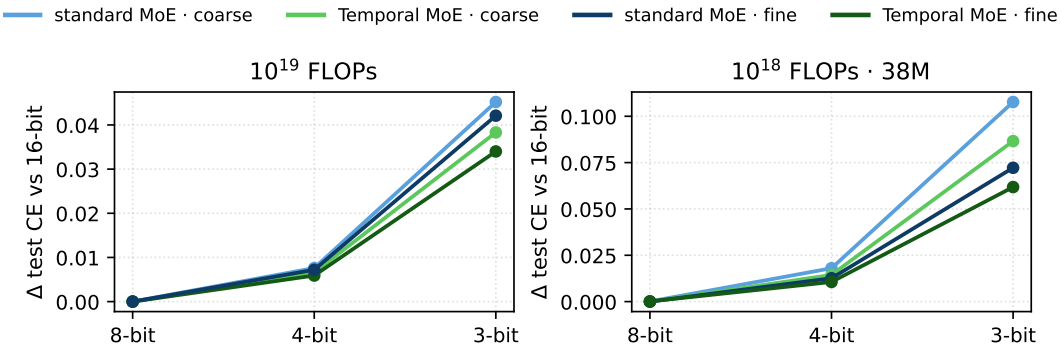


Figure 12: Test-CE cost of quantizing routed-expert weights (per-group round-to-nearest, group 128), lower is better. Panels have independent scales. The temporal model sits below its matched full MoE at every bit width.

tier much of its memory advantage because activation buffers grow with tokens in flight, and smaller ones stall it.

Table 7: Prefill micro-batch sweep at 16k context, fine-grained model. Time is prefill in ms (lower is better), memory is peak VRAM in MiB. Bold marks the operating point of Fig. 6.

	all-resident		deploy	
micro-batch	ms	MiB	ms	MiB
1024	2975	8894	5527	3159
2048	**2942**	**9512**	**4499**	**3590**
4096	2922	10746	4132	4986
16384	4723	18162	4247	13358

Where prefill time goes. Three configurations at 16k context in one micro-batch decompose the cost. The stock all-resident kernel takes 4715 ms, the expert-major kernel 4219 ms with all experts resident and 4297 ms streaming from the CPU pool. Streaming is nearly free at every context, 1.00–1.05× the resident control, so the deploy prefill cost is the kernel shape, 2.4× the stock kernel at 1k context and faster than it at 16k. Peak VRAM, with the prompt processed in 2048-token micro-batches, falls 3.7× at 1k context (2182 vs. 8174 MiB) and 2.6× at 16k (Table 7).

The vanilla-offload floor. Fig. 13 sweeps decode speed against pinned cold expert fetches per layer per token, measured with real PCIe transfers on a reconstruction of the serving models, and the rebuilt fine model reproduces the all-resident ceiling at 200.8 tok/s. Free top-k routing at active-only memory misses $k(1 - k/E) \approx 16$ experts per layer at chance overlap and about 14 at the ∼20% consecutive overlap of released MoEs, which lands at 39–43 tok/s. Points at two or fewer fetches are launch-bound and carry boost variance, and the bandwidth-bound floor is stable.

Phone protocol. The Pixel 10a runs a CPU slot pool with R expert slots per tensor in anonymous memory, evicting by releasing an expert's pages and fetching by direct reads of the expert's exact bytes before its rows are computed (UFS flash measured at 1.9 GB/s cold sequential), with four compute threads and eight fetch workers on the 1+3+4 core topology. Two gates precede every recorded number. Fetched bytes must match device-level disk counters, with agreement of 0.9 to 5%, and perplexity must be bit-identical to the pool-disabled baseline through tens of thousands of evict-and-refetch cycles. Because the all-resident model does not fit on the device, the ceiling is measured rather than inferred, on the same expert shape at the largest total expert count that does fit, run fully resident with the streaming path off and at the same context depth as the arm it divides. At $R = k = 18$, a 10.7× cut in expert memory, with turnover held at 1.05 experts per layer per token, decode measures 20.70, 17.80, 15.66, and 12.16 tok/s at context depths of 0, 1k, 2k, and 4k against ceilings of 32.42, 25.44, 20.22, and 14.58, so the fraction of ceiling climbs monotonically from 64

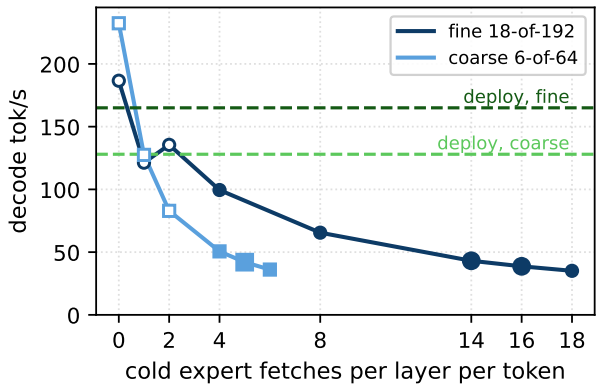


Figure 13: Decode speed against pinned cold expert fetches per layer per token. Hollow markers are launch-bound points, enlarged markers the rates behind Table 1's vanilla-offload row, dashed lines the temporal deploy rows at the same VRAM.

to 83%. Every arm passed a sustained-clock gate and a fetched-bytes gate, and a $k = 24$ shape at equal active compute moves 24% fewer bytes and reaches 88% at 4k, the same fine-granularity preference the workstation shows. Two caveats ride with these numbers. Turnover is prescribed at the design point rather than produced by a trained router, which remains the open experiment, and under that prescription residency above k is inert by construction, so an honest R curve needs the untrained-router regime instead, where decode runs 12.04 tok/s at $R = 18$ against 17.29 at $R = 36$.

Laptop protocol. We run the CPU slot-pool engine on Windows with an i7-11370H, 16 GB LPDDR4x memory, and Samsung PM981a NVMe storage. The random-weight Qwen3-MoE model has 192 experts per layer, 18 active experts, and uniform Q4_0 weights. Decode keeps only the active experts in RAM and prescribes one expert swap per layer per token, measured at 0.999. We use four compute threads and `SPLIT=1` in engine revision `43bec5f31`. Results remain provisional pending the Windows Defender exclusion record and acceptance of the Windows validation checks. Fetched bytes match the expected count exactly, and perplexity matches the baseline to every printed digit on the same binary.

Laptop speed and memory. The final 1k-context session measures 28.30 tok/s with all experts resident and 18.67 tok/s with active experts resident, retaining 66% of decode speed. These means use four baseline runs and three deployment runs. Keeping the experts in RAM while retaining our execution changes gives 19.60 tok/s, so most of the slowdown comes from those changes rather than NVMe reads. While decoding, the working set falls from 5968 to 1066 MiB, a 5.6× reduction. The earlier context sweep in Table 8 supplies the laptop curve in Fig. 6. Each ratio compares runs within that sitting. From 1k to 4k context, both CPU devices retain more of their resident baseline's speed as attention grows more expensive while expert loading per token stays fixed.

Table 8: Provisional laptop context sweep, separate from the final 1k session. Speed is in tok/s and memory is the process working set in MiB while decoding. Each depth has four all-resident runs and three active-resident runs.

Context	Resident speed	Active-only speed	Ratio	Resident memory	Active-only memory
1024	25.89	16.74	0.647	5970	1061
2048	20.42	13.83	0.677	6151	1247
4096	13.17	9.93	0.754	6511	1591

Laptop limits. During prefill this CPU implementation loads the experts a prompt needs without evicting them, so it does not yet preserve the workstation's prefill memory savings, and we report memory while decoding. The working set grows from 1061 to 1591 MiB between 1k and 4k context as the attention cache grows, which adds about the same memory to both engines. With ordinary

routing and the same expert residency, decode reaches only 0.82 tok/s at zero context, or 0.025× its resident baseline.

E INSTRUCT BENCHMARK PROTOCOL

The released models route 4 or 8 experts per token, too few to meet the bandwidth condition of §2, so these results measure accuracy under the constraint rather than stall-free serving, which would require splitting their experts.

Generation. Every configuration samples with the model's own model-card recipe, seeded, in a single pass at per-task budgets of 2048 tokens for non-thinking modes, 4096 for thinking, and 8192 for thinking IFEval. LFM2.5 thinks natively with no off mode, OLMoE has no thinking mode, and the other four models run both of their modes. An earlier truncate-and-retry ladder inflated scores by 2.6 points on average over 59 paired configurations and was retired for the single-pass protocol. During generation only the end-of-sequence token stops the model, and task stop-strings apply after the thinking block is stripped, since several fire inside chains of thought. Cells whose generations were reaching their budget were re-measured at a larger one, both arms in the same run, so every reported cell sits under 5% cap-hit on both arms.

Scoring. MMLU is dual-scored from the same generations. The stock strict filter accepts only the form "The answer is (X)" and measures few-shot format imitation, so the reported metric is relaxed answer extraction with final-answer-line priority. HumanEval is parsed channel-aware per model family for models that think in-band, and thinking text is never judged against task formats.

MBPP. All 500 test problems, scored under one producer: the task text plus its three asserts, one fenced Python block requested, thinking stripped before the last fenced block executes against the asserts. Sampling follows each model's shipped recipe at a budget of 8,192 tokens, OLMoE runs 3,328 inside its 4,096 window, and LFM2.5, which always thinks in-band, is cited at 32,768, where about 7% of items still end inside an unclosed thinking span. Binomial standard errors are 1.5–2.2 points per configuration. About two points of gemma4's and one of OLMoE's loss at $R = k$ is the model mistyping its own self-test scaffold, quantified by offline re-scores, with the scored rule unchanged.

Instruments. Full-grid configurations are single runs with binomial standard errors of 1 to 4 points. Adaptation evaluations run the full instruments, adapter-direct against each matched base configuration under one vLLM engine class; identical re-runs move a cell by about one point, HumanEval (n=164) by 2–3. Free-routing MMLU at temperature 1.0 spreads up to 3.5 points across runs, so headline MMLU numbers use multi-run means.

Adaptation training. We train with LoRA on the expert and attention tensors together with the router and norms. The 10,982-prompt pool yields 8.6M sampled tokens. The trainer hosts an in-process vLLM engine that sleeps between refreshes. Every 16 optimizer steps the current adapter is synced into it bit-exactly, and 256 fresh responses are sampled under the R=8 bound with each model's card recipe (temperature 0.7, top-p 0.8; Qwen3.5 adds its presence penalty of 1.5). The reverse KL is computed analytically over the teacher's top 50 tokens with the tail renormalized, at temperature T=2 with the loss scaled by T^2. Learning rates are 10^{-4} for gemma4 and 3×10^{-5} for Qwen3.5, each saturated in its own sweep; KL temperature, sampling temperature, refresh cadence, and a doubled token budget left GSM8K at R=8 within noise of where the learning rate put it, and the sampled-token KL trace never predicted the evaluation. A refresh costs 55 to 70 seconds on both models. Matching Qwen3.5's cycle time to gemma4's required the `flash-linear-attention` fast path in the trainer, a 0.65 GPU-memory share for the engine to avoid KV preemption, and a persistent-mask implementation of the presence penalty that is output-identical to vLLM's native path at 1.8× its speed. The full-pool budget is one pass over the prompts; Table 9 gives the pool's composition. A math-heavy alternative (the d7 pool, this pool without the code lane, at 3.4M sampled tokens) buys about two more GSM8K points at R=8 on both models (+5.2 gemma4, +6.5 Qwen3.5) and gives them back on MMLU, HumanEval, and MBPP.

lane	n	share	source
domain8k	4,958	45.1%	WildChat, general instruction without code
codelane	2,500	22.8%	WildChat, code requests
mathlane_v2	2,341	21.3%	StackMathQA, real math question answering
d5_fewshot	1,183	10.8%	OASST2, few-shot formatted

Table 9: The 10,982-prompt adaptation pool. Every lane is 8-gram-screened against the test sets; the d7 alternative of the prompt-mix ablation is this pool without the code lane.

The thinking-on adapters use the same recipe and budget with thinking enabled during sampling (rollout cap 8192) and are evaluated with thinking on at §6's budgets against full-instrument thinking-on baselines.

Table 10: The Table 2 adapters, trained under R=8, evaluated at the fraction-matched 12.5% setting (R=16 gemma4, R=32 Qwen3.5), score changes against the unadapted base under the same evaluation setting.

	arm	GSM8K	IFEval	MMLU	HumanEval	MBPP	mean_5
gemma4	R=16	86.4 (-0.2)	86.9 (-0.9)	93.9 (+1.3)	97.6 (+1.2)	89.4 (-0.2)	$+0.2$
Qwen3.5	R=32	85.1 (+5.3)	84.3 (-0.9)	91.2 (-2.2)	92.1 (+3.0)	80.8 (+1.8)	$+1.4$

Table 11: Thinking-on adapters, trained with the Table 2 recipe, evaluated with thinking on over every benchmark item at R=8, absolute scores with adapted-minus-base deltas in parentheses.

	GSM8K	IFEval	MMLU	HumanEval	MBPP	mean_5
gemma4	85.9 (+1.1)	94.3 (+2.6)	93.4 (-0.9)	98.8 (+4.3)	86.2 (+8.4)	$+3.1$
Qwen3.5	85.7 (+0.6)	79.9 (-2.8)	91.7 (+4.4)	93.3 (+6.7)	82.4 (+2.0)	$+2.2$

One engine class per comparison. vLLM loads Qwen3.5's raw checkpoint under its multimodal class and merged checkpoints under a text-only class. With bit-identical weights the two disagree at the bf16 level, enough to flip 219 of 1319 GSM8K items at R=8 and read about 1.2 points in the merged arm's favour, so adaptation numbers are adapter-direct on the raw class for base and adapted models alike.

WritingBench. The official queries and critic model run locally through the same constrained serving stack, three disjoint 50-query English subsets per configuration, scored 1 to 10 by the critic, with deltas paired within subset (paired standard deviations 0.05 to 0.15).

Damage tracks substitutable capacity. Self cross-entropy (self-CE), each model's cross-entropy on its own frozen responses, isolates the constraint from benchmark harnesses. At the matched 12.5% fraction the damage runs from 0.94 nats per token for OLMoE down to 0.11 for Qwen3.5 (Table 12). The ordering favors models with more routed experts and an always-resident shared expert, and the model furthest out of line with pure expert count, OLMoE, is the one whose free router leans hardest on token identity when probed as in §4. This is the released-model face of the pretraining picture, where fine granularity tolerated the constraint best, and it points the same direction the field is already moving, toward more and smaller experts.

Tolerance is predictable before any benchmark runs. A natural guess is that models whose router probabilities move most under the mask suffer most. The opposite holds, with router-distribution displacement anti-ordering benchmark damage across the six models (Spearman -0.49). What orders it is functional displacement, the relative output error of the routed-expert layer when the resident set substitutes for the free choice on identical inputs, at Spearman $+0.66$ and exactly ordering damage within each residency fraction (Fig. 14). Substitutability of expert function, not stability of routing probabilities, is what tolerance to the constraint measures, which fits §4's finding that the constraint changes how experts are used far more than what they compute.

Table 12: Self-CE damage on each model's own frozen responses (nats per token, constrained minus free routing, lower is better), at each model's $R = k$ and at the matched 12.5% fraction. The nine-fold spread at matched fraction favors models with more routed experts.

model	k/E	at $R = k$	at 12.5%
OLMoE-Instruct (64 experts, $k = 8$)	12.5%	0.94	0.94
LFM2.5-A1B (32, $k = 4$)	12.5%	0.31	0.31
gemma4-26B-IT (128, $k = 8$)	6.25%	0.38	0.21
Qwen3.5-35B (256, $k = 8$)	3.1%	0.21	0.11

Displacement measures. For the predictor above, router displacement is the mean per-token total-variation distance between the router distribution computed free and under the constraint on the same inputs, and functional displacement is the relative output error of the routed-expert layer when the resident set substitutes for the free choice, $\|y_{\text{masked}} - y_{\text{free}}\|/\|y_{\text{free}}\|$, averaged over layers. Figure 14 plots both against mean benchmark damage at $R = k$. Router displacement anti-orders damage (Spearman -0.49) while functional error orders it ($+0.66$) and orders it exactly within each residency fraction. LFM contributes its native thinking damage, having no off mode.

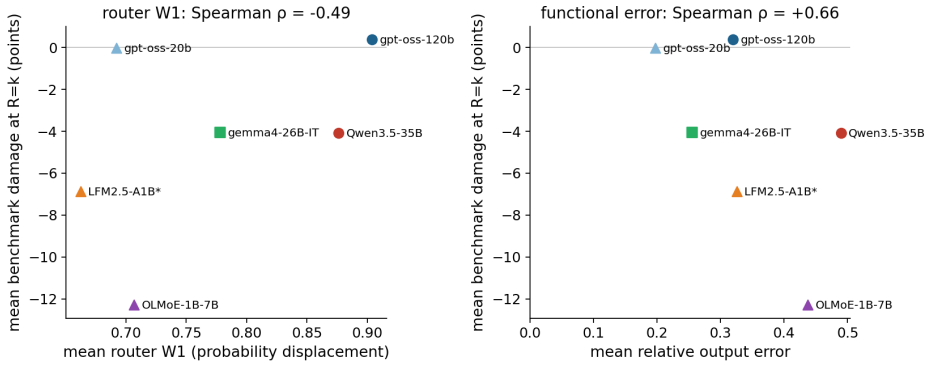


Figure 14: Mean benchmark damage at $R = k$ against router probability displacement (left) and functional displacement (right), one point per model, marker by residency fraction. The router-space measure anti-orders damage, the function-space measure orders it.

F Length analysis details

Data and definitions. Length measurements pair the same items free and constrained by document id, 200 items per configuration and 164 or 228 on the code and knowledge harnesses. Length means the whole generation, thinking block plus answer. Recovering it is not uniform across the grid: the dumps record the post-strip answer and the thinking segment in separate fields, and which field carries the total depends on when the dump was written, so the route is chosen per dump and validated against the records that store the total directly (agreement on 100% of the 3,800 items carrying both, median ratio 1.000). The rule is documented with the capture code that writes the fields. A cap-hit is a generation within 8 tokens of its cell's declared budget, never its observed maximum, since a single over-cap generation moves a maximum-based reference past the pile-up at the budget and hides it. A long generation is one running more than twice its cell's free-routing median, a threshold fixed on the free arm and applied to both, so the two arms' counts are comparable.

The constraint stretches the tail and leaves the middle alone. Paired per-item token counts, thinking block and answer together, cover all five benchmarks (App. F). Under the tightest cache the median generation runs 1.03 to 1.08× its free-routing counterpart, so the grid mean of 1.05–1.14× understates a distribution that is flat through the middle and heavy at the top. The 95th percentile of the per-item constrained-to-free length ratio is 1.9–2.5× and the 99th 2.8–4.6× (Table 13). The effect belongs to thinking, since with thinking off the median is exactly 1.00 and the 95th percentile only

1.5 to 1.6, and WritingBench, the writing benchmark and the only one every model runs thinking-off, does not lengthen at all at 0.99. What grows is deliberation rather than answer, and on MMLU the answer shortens to 0.99 while the thinking block grows to 1.06.

Length is not what the constraint charges for. Generations longer than twice their cell's free-routing median are wrong 1.5–4.7× as often as normal-length ones, and 1.8–7.3× as often under free routing, so the long ones were already the risky ones. Residency adds few of them, and those it does add carry 6–25% of the loss on math, instruction following and knowledge, against 52% on code (App. F). Everywhere but code the loss is spread across generations of every length, which is the signature of a per-token substitution error rather than a few derailments, and it is the same functional displacement that orders damage across models in §6.

Percentiles. Table 13 gives the per-item ratio of constrained to free-routing length at each percentile, pooled within each surface. The thinking configurations are flat through the median and heavy at the top, while the non-thinking ones stay near 1.00 until the last few percent. The one MMLU row without thinking is a single configuration and is reported for completeness rather than as a trend.

Table 13: Per-item constrained-to-free generation length at each percentile, thinking block and answer together. Values above 1 are longer under residency. The last column is the token-weighted mean over all items, which the tail pulls far less than the percentiles suggest.

surface	cells	items	P50	P75	P90	P95	P99	mean
thinking on								
GSM8K	14	2,800	1.05	1.27	1.60	1.91	2.84	1.07
IFEval	14	2,800	1.03	1.32	1.84	2.35	4.45	1.07
HumanEval	14	2,296	1.08	1.34	1.76	2.14	3.76	1.14
MMLU	14	3,192	1.03	1.36	1.90	2.48	4.56	1.05
thinking off								
GSM8K	5	1,000	1.00	1.12	1.33	1.62	3.58	1.02
IFEval	5	1,000	1.00	1.11	1.31	1.53	2.59	1.00
HumanEval	5	820	1.00	1.11	1.66	2.52	4.11	1.07
MMLU	1	228	1.34	2.48	4.40	6.05	9.51	1.52
WritingBench	8	1,200	1.00	1.06	1.19	1.36	1.87	0.99

How a generation ended. Fig. 15 splits every constrained generation by how it terminated and reports how often each group was wrong. Generations that reached their budget emitted no usable answer and are wrong mechanically, so they form one group rather than being split by whether the thinking block closed. They are 1.0 to 1.5% of items on the accuracy benchmarks, small enough that truncation no longer carries the damage. The group that speaks to length is the one that ran past twice its free counterpart and still finished, at 1.2–3.7× the normal-length error rate, worst on code. WritingBench is critic-scored rather than marked right or wrong, so it carries mean score drop on its own axis, and its budget-reaching share of 16.1% is the one place in the grid where truncation is still doing real work.

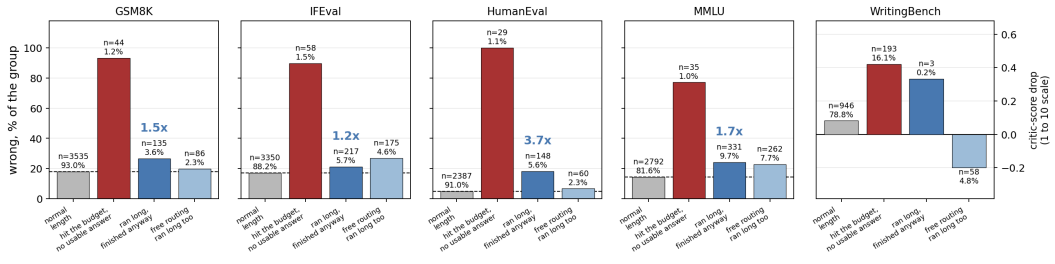


Figure 15: How a constrained generation ended against how often it was wrong, pooled over the reported cells. Dashed line is the normal-length rate; the multiplier over it is marked on the long-but-finished group.

Where the extra tokens sit. Fig. 16 separates each configuration's mean length change into the part carried by generations at their budget and the part carried by generations that never approach it. The two are uncorrelated across configurations ($R^2 = 0.00$), so a cell can lengthen broadly, at the wall, or both, and the flip strip beside it shows the length change of items whose correctness moved.

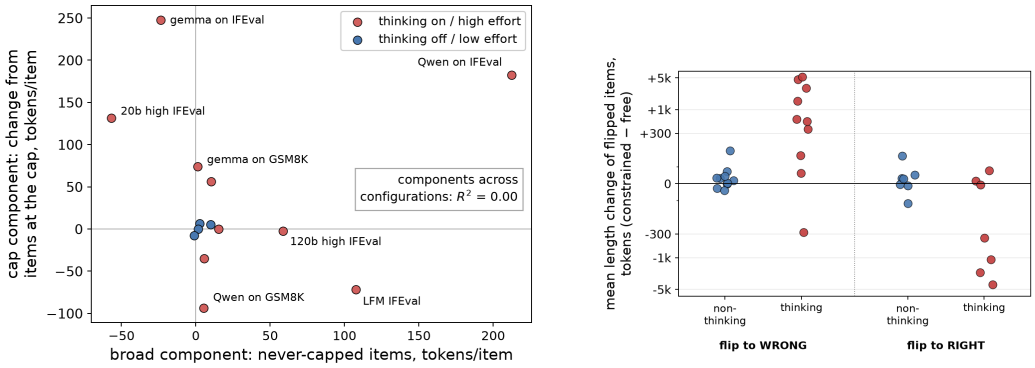


Figure 16: **Left,** the two components of each configuration's mean per-item length change (paired document ids, 200 items per configuration), cap-carried against never-capped. Configurations occupy every quadrant and the components are uncorrelated ($R^2 = 0.00$). **Right,** mean length change of flipped items, one point per (model, task) configuration with at least five flips, red thinking modes, blue non-thinking. Flips to wrong mostly grew, rescues are avoided free-routing blow-ups, non-thinking flips barely move.

Per-configuration detail. Table 13 pools within a benchmark, which hides how differently models reach the same pooled tail. Figure 17 unpools it for the thinking block alone, one row per thinking configuration. gemma4's growth sits in the tail (P95 ratio 1.86 on GSM8K), gpt-oss-120b's at the median, and gpt-oss-20b shortens at high effort. Ratios in configurations whose free-routing median thinking runs under about 100 tokens are a few tokens of noise.

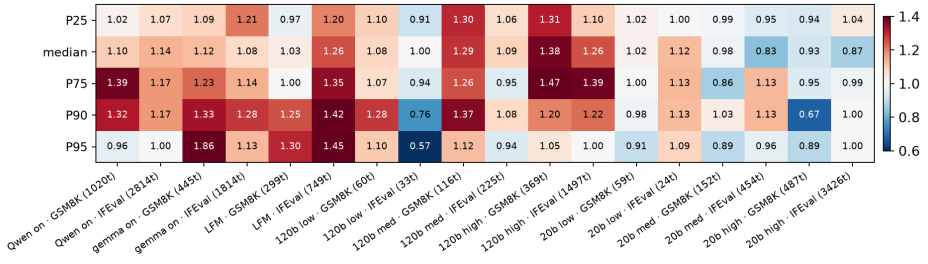


Figure 17: Think-token ratio, tight cache over free, at five percentiles for every thinking configuration. Red is longer under the constraint, blue shorter, and each label carries the free-routing median think length in tokens.

What carries the damage. Splitting each benchmark's rise in error rate into the part explained by residency producing more long generations and the part explained by generations of every length becoming more error-prone puts the second at 75% on GSM8K, 89% on IFEval and 94% on MMLU, against 48% on HumanEval. Long generations themselves are wrong 1.5–4.7× as often as normal-length ones under the constraint and 1.8 to 7.3× as often under free routing, so the association between length and error is present in both regimes and residency shifts the count rather than the per-generation cost. That count moves little except on code, where the long share rises from 10.6 to 15.4%. Flip traffic tells the same story from the item side: across the thinking configurations the constraint flips more items to wrong than it rescues while leaving the number of long generations per setting nearly unchanged.

The length-damage null. Fig. 18 plots damage at the tightest setting against free-routing response length across tasks and models. No significant relationship survives the corrected protocol (Spearman -0.18, $p = 0.63$), and the earlier protocol's significant correlation came from budget-truncated short-answer configurations and reconstructed lengths.

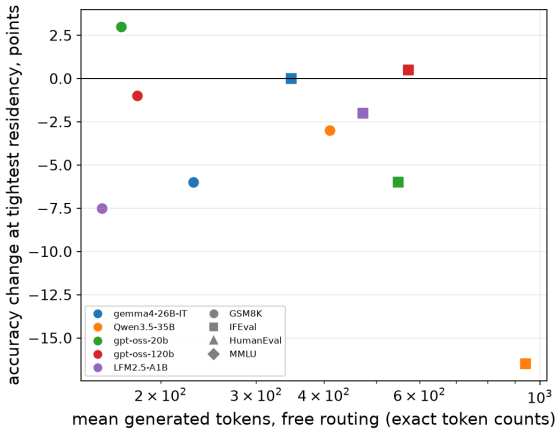


Figure 18: Damage at the tightest residency setting against mean free-routing response length, color per model, marker per benchmark, exact token counts.

Adaptation and length. Figure 19 normalizes each bar by the released checkpoint's generation length under free routing. This reference separates changes caused by residency from changes in the adapted model's own free-routing behavior. The thinking-on panels pair each released checkpoint with its thinking-on adapter. The thinking-off panels use the adapters from §7. With thinking disabled, the released models lengthen only on HumanEval, by factors of 1.31 for Gemma4 and 1.19 for Qwen3.5. Lengths remain flat elsewhere, including WritingBench. Adaptation reduces Gemma4's excess length from 31% to 14%, while Qwen3.5's adapted responses on HumanEval are 64% longer than the original baseline even under free routing.

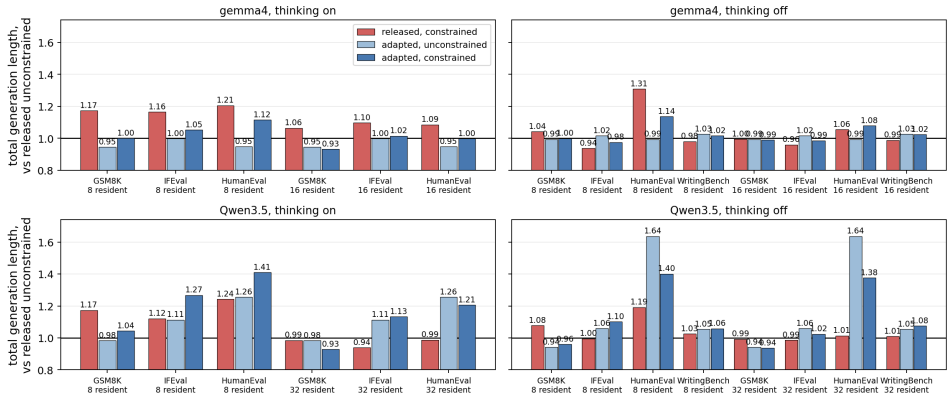


Figure 19: Total generation length under residency, thinking block and answer together, every bar divided by its panel's reference, the released model under free routing. The middle bar of each group is the adapted model's own free-routing length.