Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

# Predatory MoE: Serving Mixture-of-Experts with a Sequence-Local Expert Roster

**ChaeWoo Son**

soncwnetp@gmail.com

Independent Researcher https://orcid.org/0009-0009-6108-2373

**Research Article**

**Keywords:** mixture-of-experts, expert offloading, sparse models, efficient inference, routing, memory-constrained serving

**Posted Date:** August 19th, 2026

**DOI:** https://doi.org/10.21203/rs.3.rs-10741449/v1

**License:** This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

**Additional Declarations:** The authors declare no competing interests.

# Predatory MoE: Serving Mixture-of-Experts with a Sequence-Local Expert Roster

ChaeWoo Son
Independent Researcher
soncwnetp@gmail.com

**Abstract**

Open-weight mixture-of-experts (MoE) models activate only a small fraction of their weights per token, yet serving them in memory-constrained settings typically requires keeping every expert resident, so memory follows total rather than active parameter count. Offloading inactive experts reduces residency, but when routing scatters across many experts, weights must be fetched nearly every token. We study whether routing can be made *sequence-local* — each sequence served by a small, fixed roster of experts — so that a roster can be pinned once per sequence and decoding can proceed with few further transfers. On a controlled MoE testbed, we first examine whether sequence-locality can be instilled as a training objective, and find that it is not persistent: selection biases concentrate routing only when applied from early training; continued training without the bias restores scattered routing; and a natural distillation variant that trains the router to imitate its own biased selections is self-referential, and in our experiments destabilized routing. Based on this analysis, we instead treat sequence-locality as a routing rule applied identically during training and serving: a per-sequence roster of $m$ experts per layer, formed over a short prefix and then fixed. Under this rule, per-sequence working sets remain close to $m$ throughout training, and a 600-step fine-tune under the deployed rule adapts a pretrained testbed MoE to match training under the rule from scratch: +0.9% perplexity at a 50% reduction in resident expert weights, and +5.0% at 75% (with 8 and 32 experts respectively). In trace-driven serving simulation with matched memory budgets, the resulting demand pattern reduces expert transfers by up to 32× relative to an LRU cache and recovers 94% of full-residence decode throughput with one quarter of the expert weights resident. Finally, we observe that, in our testbed, unconstrained training gradually erodes how cheaply a model can be served under a pinned roster even after perplexity converges — suggesting potential benefits to exposing serving constraints during training.

## 1 Introduction

Sparse mixture-of-experts (MoE) architectures now dominate open-weight language model releases at the largest scales, and some recent flagships reach trillions of total parameters while activating only a few percent per token (e.g., 2.4T total with ∼95B active). Sparsity thereby decouples capability from per-token compute [1]. It does not, by itself, decouple capability from serving memory: whenever all experts must share one memory pool — the default for single-accelerator, workstation, and local deployments — the footprint follows the *total* parameter count even though each token touches only the *active* fraction. Multi-device expert parallelism sidesteps this by sharding experts across a cluster, but presumes a cluster; our focus throughout is the memory-constrained, single-stream setting, where a model can be open in license yet remain out of reach in practice.

Expert offloading is the standard response in this setting [2]: keep a bounded set of experts resident, park the rest in host memory or on disk, and fetch on demand. Existing systems then adapt to the routing demand as they find it — caching recently used experts, prefetching predicted ones. The demand itself, however, is hostile to any bounded cache: token-level routers may select any expert at any token, and in our 32-expert testbed an unconstrained model touches on average 28 of 32 experts per 256-token sequence at every layer. We ask a question complementary to the systems work: not how to serve a given demand pattern, but whether the demand pattern can be changed. Concretely, can routing be made *sequence-local* — each sequence served by a small roster of experts, fixed early and rarely revisited — so that resident memory follows a roster budget rather than the total expert count?

A natural first answer is to train sequence-locality into the model, and we begin there. On a controlled MoE testbed we apply a rich-get-richer selection bias — experts already used within a sequence are favored for its remainder, constructed in the manner of auxiliary-loss-free balancing biases [3] but with the concentrating sign — and while applied it behaves as intended: per-sequence working sets shrink substantially at under 1% perplexity cost. Three observations, however, indicate that in our setting the resulting locality is a property of the routing *procedure*, not of the learned *weights*. First, the bias reshapes routing durably only when present from early training; applied late, it yields an immediate selection-time effect and no measurable further adaptation within our training horizon. Second, the effect reverts: if the bias is switched off midway, continued unbiased training restores the original scattered routing in a comparable number of steps. Third, a natural attempt to internalize the bias — distilling the router toward its own biased selections — has a moving, router-dependent target; in our experiments this self-reference destabilized routing rather than anchoring it.

Taking these results at face value, we stop asking the model to *learn* sequence-locality and instead impose it as a *routing rule*, applied identically at training and serving time — the status already held by top-$k$ selection and by balance biases, which no one expects a router to internalize. The rule: each sequence maintains, per layer, a roster of the $m$ experts most used so far; once the roster fills, non-members face a fixed selection barrier; at serving time the roster is pinned after a short prefix. The budget $m$ is a deployment parameter read off the available memory, not a regularization strength to be tuned. Two properties distinguish this from expert pruning: rosters differ across sequences, so the model's full expert capacity remains in use globally (expert load stays balanced in our runs), and no weights are removed — only their per-sequence residency is scheduled. In our testbed the rule holds per-sequence working sets at $m$ across full training runs, without the drift or collapse we observe for its amplifier-style predecessor.

Because the rule lives in the routing procedure, it can be attached to a model pretrained without it — the practically important case. Attached at inference alone, it is costly: in our testbed, pinned-roster perplexity rises 32% over the equal-step baseline with 8 experts and roughly 9× with 32, with high variance across seeds — consistent with pretrained routers' token-level preferences conflicting with a sequence-level constraint, though we do not claim to have isolated the mechanism. A short fine-tune *under the deployed rule* — 600 steps, 20% of our pretraining budget — removes nearly all of this cost: the retrofitted model matches one trained under the rule from scratch, at +0.9% perplexity for a 50% reduction in resident expert weights and +5.0% for 75% (32 experts), relative to an equal-step baseline control. We also observe the converse in our testbed: training *without* the rule steadily increases the later cost of serving under one, even after perplexity has converged — evidence, within our controlled setting, that serving constraints may be worth exposing during training.

Finally, we translate the routing change into serving terms by replaying real routing traces through a cache simulator. The comparison is at matched memory budgets against an LRU cache

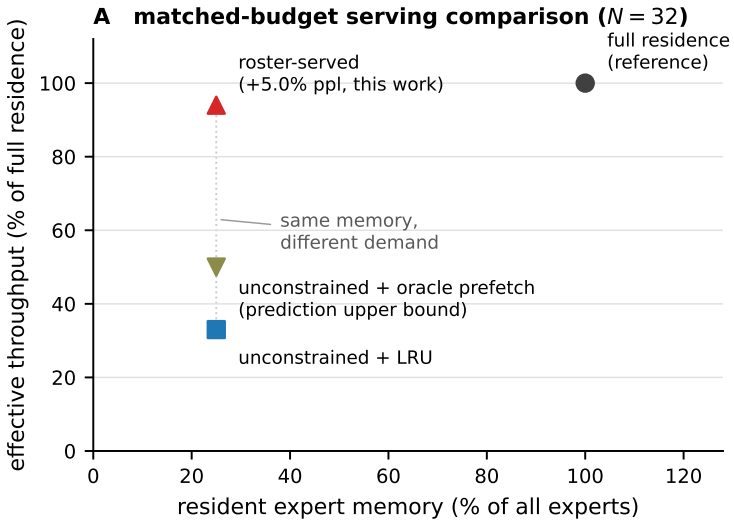



**Figure 1: One quarter of the expert memory retains 94% of full-residence throughput on our testbed.** Memory–throughput–quality summary ($N$=32; trace-replay simulation under the accounting of §6). At matched memory (25% of expert weights resident), roster serving recovers 94% of full-residence effective throughput at +5.0% perplexity, where LRU serving of the unconstrained model reaches 33%. Throughput percentages inherit assumed system constants; transfer counts do not (§6).

— the natural policy for unconstrained demand — and charges our method's roster prefetch in full, with no compute overlap assumed. Under these conservative terms, the roster-served model reduces expert transfers by up to 32× and recovers 94% of full-residence decode throughput with one quarter of the expert weights resident. In the limiting case $m = k$, resident expert weights equal the model's active-parameter budget — the floor this approach makes reachable.

**Contributions.** (1) A controlled empirical analysis of why sequence-local routing resists being instilled as a training objective: timing sensitivity, reversion under continued training, and the self-referential failure of router distillation (§2). (2) A roster-budget routing rule with a deployment-interpretable budget $m$ and, in our runs, stable training dynamics (§3). (3) A retrofit recipe — brief fine-tuning under the deployed rule — matching from-scratch training on our testbed, plus the observation that unconstrained training erodes serveability (§4–5). (4) A trace-driven serving analysis under matched budgets and conservative accounting (§6).

**Scope.** We are deliberate about what this paper does and does not show. All quantitative results come from a synthetic, single-domain testbed of up to 32 experts — chosen not for convenience but because it lets us switch the locality pressure on and off cleanly and attribute effects to it. Two stressors are absent by construction and remain untested: batched multi-stream serving (where resident memory follows the union of rosters) and production-scale models and corpora. A third — sequences that change topic midstream, a known hazard for any pinned-roster scheme — we test in a first synthetic form (§5.4); sharper, real-text shifts remain untested. §7 states the arithmetic for extrapolating to large open-weight MoEs together with each unverified assumption it relies on; we present that section as motivation for future validation, not as a claim. Figure 1 summarizes the memory–quality–throughput trade-off.

| Condition | PPL | WS (of 8) | $WS_0$ | $H_{glob}$ (max 2.08) |
|---|---|---|---|---|
| base | $4.455 \pm 0.004$ | $7.77 \pm 0.05$ | — | 2.04 |
| amplifier, early $\lambda$=1 | 4.459 | $5.77 \pm 0.11$ | — | 2.02 |
| amplifier, early $\lambda$=2 | $4.464 \pm 0.004$ | $5.18 \pm 0.09$ | 5.28 | 1.98 |
| amplifier, early $\lambda$=4 | $4.493 \pm 0.010$ | 4.58 | — | 1.89 |
| amplifier, late $\lambda$=1 | 4.460 | $6.64 \pm 0.20$ | — | 2.03 |
| bias removed midway (§2.4) | 4.454 | $7.15 \pm 0.06$ | = WS | 2.03 |
| + distillation (§2.5) | $4.553 \pm 0.022$ | $3.21 \pm 0.33$ | 3.55 | $1.60 \pm 0.14$ |
| + self-distillation, bias removed | 4.494 | $5.19 \pm 0.33$ | — | 1.85 |

**Table 1:** Amplifier-bias conditions ($N$=8, 3 seeds, mean±std where recorded; dashes denote values we did not log for that run). $WS_0$ evaluates the same checkpoint with the bias off. PPL differences are small on this task by construction (§2.1); the distillation row is the exception.

# 2 Can Sequence-Locality Be Trained In?

## 2.1 Testbed

We conduct all experiments on a small decoder-only MoE trained on synthetic multi-domain data, chosen so that the locality pressure can be switched on and off cleanly and its effects isolated. The model has 4 layers, $d = 256$, and one MoE block per layer with $N \in \{8, 32\}$ experts and top-2 routing; gates are computed from the pure router distribution while all biases act on selection only, following the loss-free balancing design [3]. A sign-feedback balance bias maintains global load balance in all conditions. Data are three synthetic Markov domains over a shared vocabulary, one domain per sequence ($T = 256$); this gives each sequence a well-defined "home" expert demand while keeping the corpus balanced. We report, over 3 seeds: perplexity (PPL); the per-sequence working set WS, the number of distinct experts a sequence selects per layer; global selection entropy $H_{glob}$ (balance); and, where relevant, the same metrics with the locality mechanism switched off at evaluation, denoted $WS_0$. Unless noted, $N = 8$; §5 repeats key results at $N = 32$. We emphasize that PPL on this task is largely insensitive to routing (all conditions below fall within 4.44–4.50), so PPL serves as a guardrail rather than a discriminating signal; the discriminating signals are WS and, later, pinned-roster perplexity.

## 2.2 An amplifier bias concentrates routing while applied

Our training-based candidate is a rich-get-richer selection bias: at token $t$, expert $i$ receives a selection-score bonus $\lambda \cdot (\bar{c}_{i,t} - 1/N)$, where $\bar{c}_{i,t}$ is the expert's average routing probability over the sequence so far. This is the auxiliary-loss-free balancing update transplanted from the batch level to the sequence level, with the concentrating sign, and like it, it touches selection only — no gradients are altered. Applied from the start of training ("early"), it works as intended while active: at $\lambda = 2$, WS drops from $7.77 \pm 0.05$ to $5.18 \pm 0.09$ of 8 experts at +0.2% PPL, and $\lambda = 4$ reaches 4.58 at +0.9% (Table 1). Larger $\lambda$ shows early signs of strain — global balance entropy declines and per-seed trajectories become non-monotone — which we revisit in §2.5.

## 2.3 Timing: late application yields no measurable adaptation

If the bias trains locality into the weights, applying it later in training should still work, merely with less time. It does not, in our runs: annealing the same bias in mid-training produces a step change in WS at the moment the bias reaches full strength ($7.4 \to 6.7$ at $\lambda = 1$) and then no further

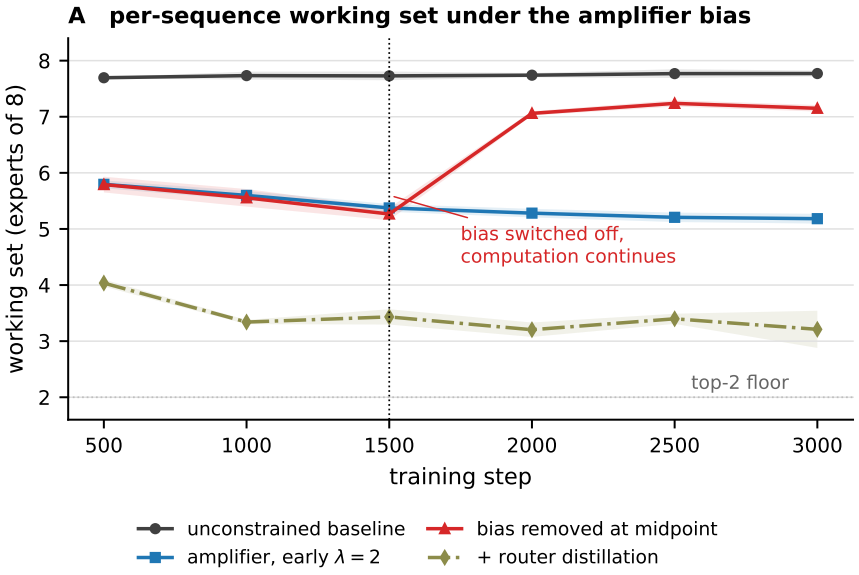



**Figure 2: Trained-in locality reverts once the bias is removed.** Per-sequence working set over training ($N$=8, mean over 3 seeds). Locality installed by the amplifier bias ($\lambda$=2, applied from step 0) reverts toward the unconstrained baseline within ∼1,500 steps once the bias is switched off at the midpoint; adding router distillation instead drives WS toward the top-2 floor. Curves are from our testbed and single task; §9 discusses settings where these dynamics could differ.

movement over the remaining 1,500 steps — WS stays within 0.02 of its post-activation value. The early-applied counterpart at equal $\lambda$ reaches $5.77 \pm 0.11$, a gap of roughly one expert that the late run never closes. Whatever the bias contributes beyond its immediate selection-time effect appears to be acquired only while routing and experts are still co-forming.

### 2.4 Reversion: continued training erases the effect

The sharper test is to train with the bias and then remove it. Evaluated with the bias switched off, an early-$\lambda$=2 model still routes locally ($\mathrm{WS}_0 = 5.28$ vs. WS = 5.18): at any fixed checkpoint, the concentration does reside in the weights. But it does not survive further training. Switching the bias off at the midpoint and continuing otherwise unchanged, WS climbs from 5.3 back to $7.15 \pm 0.06$ within 1,500 steps — most of the way to the unconstrained 7.77 — while PPL is indistinguishable from baseline throughout (Fig. 2). We read this as the training objective itself preferring scattered routing, if only weakly: on this task the language-modeling loss is flat across routing patterns (§2.1), and absent the bias, optimization drifts back along that flat direction. We do not claim this drift is universal; capacity-constrained settings, where specialization pays in loss, could behave differently. In our setting, however, locality installed by the bias has the character of a displaced equilibrium, not a new one. The reversion also survives a fourfold reduction in expert width ($d_{\mathrm{ff}}$ 512 → 128): at the reduced width the biased run's experts co-adapt strongly enough that switching the bias off at a fixed checkpoint costs +52% perplexity, yet under continued unbiased training routing and experts revert *together*, ending at baseline perplexity and the same WS = 7.15 as before. Final language-modeling loss remained routing-flat even at the reduced width, so this replication narrows, but does not close, the capacity-regime caveat of §9.

### 2.5 Distillation is self-referential and, in our runs, destabilizing

A standard remedy would be to make the weights own the behavior: add a distillation term that trains the router (and only the router — inputs are detached) to imitate the biased top-$k$ selections. The difficulty is structural: the target is a function of the router being trained. The bias sharpens selection; distillation moves the router toward the sharpened selection; the sequence-average $\bar{c}$ that drives the bias sharpens in turn. In our runs this loop does not find a fixed point. With bias and distillation active together, WS slides to $3.21 \pm 0.33$ — near the top-2 floor — while global balance entropy collapses from 2.04 to $1.60 \pm 0.14$ and PPL degrades by +2.2%, our only quality failure in this section (Table 1). Halting the bias midway while keeping distillation ("self-distillation") merely slows the reversion of §2.4: the target now tracks the router's own drifting selections, and WS climbs $4.2 \rightarrow 5.2$ over the second half without stabilizing. We tested one distillation form at one coefficient; gentler variants may exist. But the moving-target structure is not an artifact of our choice of coefficient, and we found no setting that both preserved quality and anchored locality.

### 2.6 Takeaway

Across these probes, sequence-locality behaved as a property of the routing procedure: present while the procedure enforces it, timing-sensitive to install, reverting when enforcement stops, and resistant to being transferred into the weights by self-imitation. Rather than continuing to engineer persistence — heavier distillation, frozen teachers, auxiliary objectives — we take the observations at face value and change the goal: if the procedure is what carries the behavior, let the procedure be the mechanism, permanently. The next section designs that mechanism so that its equilibrium is explicit rather than emergent.

## 3 Sequence-Locality as a Routing Rule

### 3.1 Design choices motivated by §2

The observations of §2 motivate — though, drawn from one controlled setting, they do not by themselves compel — three design choices. Having watched trained-in locality revert under continued training (§2.4), we choose a mechanism that does not depend on the weights retaining the behavior: we place it in the routing procedure and keep it active at both training and serving time. This is a conservative choice, not a claim that no persistent training-based scheme exists; it puts the mechanism in the same category as top-$k$ selection and balance biases, which are specified as procedures rather than learned. Having watched rich-get-richer amplification strain at large $\lambda$ and run away under distillation (§2.5), we choose a mechanism with an explicit equilibrium rather than an emergent one: the amplifier has no setpoint, so where it stops is a balance of $\lambda$, the balance controller, and training dynamics — a balance we observed failing. And the timing result (§2.3) suggests what training remains useful for under such a mechanism: not installing the behavior, but letting experts co-form beneath it. The rule below realizes these choices.

### 3.2 The roster rule

Each sequence maintains, at every MoE layer, a running count $c_i$ of how often expert $i$ has been selected so far. The roster $R$ is the top-$m$ experts by count. Selection proceeds in three phases. *Exploration:* until $m$ distinct experts have been used, no bias is applied, so that the roster is filled by the router's own preferences rather than fixed at the first $k$ picks. (In our testbed this guard was necessary: without it, the first token's selections self-reinforced and the roster collapsed to

| Condition ($N$=8) | PPL | WS | $\text{PPL}_{\text{mask}}$ ($m$=4) | $H_{\text{glob}}$ |
|---|---|---|---|---|
| base | $4.455 \pm 0.004$ | $7.77 \pm 0.05$ | — | 2.04 |
| roster rule, from scratch | $4.467 \pm 0.005$ | $4.18 \pm 0.03$ | $4.470 \pm 0.005$ | 1.94 |

**Table 2:** Training from scratch under the roster rule ($m$=4, $\delta$=0.3; 3 seeds). The working set holds at the budget across the full run, and pinned-roster perplexity is within noise of unconstrained perplexity for this model.

$k$ at initialization.) *Barrier:* once the roster is full, members receive a selection bonus $+\delta$ and non-members $-\delta$, so that entry requires an outside expert to overcome a $2\delta$ score deficit (up to the balance bias, which remains active); on entry it displaces the least-used member. Members share the same bonus, so competition inside the roster is decided by the router scores alone. *Pinning (serving):* at deployment, the roster is frozen after a prefix fraction $p$ of the sequence ($p = 0.2$ throughout) and selection is restricted to it. In the serving design we simulate in §6, each pinned roster entry maps to one expert held resident, and each pre-pinning barrier crossing to one prefetch; other serving designs are possible. As in §2, all of this acts on selection only; gates remain the pure router distribution.

### 3.3 What the parameters mean

$m$ is the roster budget: the number of experts per layer a deployment keeps resident. Because residency memory scales linearly in $m$, a deployment can start from its memory limit and read off a candidate $m$; what that choice costs in quality is an empirical question, which §5 answers with cost-versus-$m$ curves rather than a single recommended value. $\delta$ is the entry barrier, setting roster rigidity between soft preference (small $\delta$) and hard constraint ($\delta \to \infty$); we use $\delta = 0.3$ on probability-scale scores unless noted and examine its behavior — including a regime where larger is not better — in §4. Top-$k$ routing forces $m \geq k$, and at the limiting budget $m = k$ the resident *routed-expert* weights match their share of the per-token active parameters; always-active components (attention, embeddings, any shared experts) are resident in every scheme and unaffected by ours. The rule's runtime state is one integer counter per expert per layer, negligible beside the expert weights it schedules.

### 3.4 The rule is stable where the amplifier was not

Trained from scratch under the rule ($N = 8$, $m = 4$, $\delta = 0.3$), the working set sits at $4.18 \pm 0.03$ across the entire run — within 0.2 of the budget, the excess coming from exploration and occasional barrier crossings — with no drift over 3,000 steps, global balance entropy at 1.94/2.08, and PPL at +0.27% over baseline (Table 2). The contrast with §2.5 is the point: given a setpoint, the same concentrating pressure that ran away under amplification parks at the budget. Pinned-roster evaluation costs essentially nothing for this model (within 0.1% of its own unpinned perplexity, +0.3% over the base model): a model that grows up under the rule rarely wants an expert outside its roster. We note two things this rule is not. It is not pruning: rosters differ across sequences, every expert serves some sequences, and global load remains balanced in our runs. And it does not make the model *learn* locality — evaluated with the rule off, $\text{WS}_0$ rises to 6.9, and we expect (though have not run) the reversion of §2.4 to apply if training continued without the rule. By construction, the rule does not ask the weights to retain anything.

### 3.5 Serving cost, previewed

Under pinning, quality has a precise operational meaning: pinned-roster perplexity, $\text{PPL}_{\text{mask}}$, the perplexity of decoding the post-prefix remainder with selection restricted to the pinned roster. This is the price a deployment actually pays for keeping only $m$ experts resident, and it is the primary metric of the rest of the paper. For the from-scratch model above, $\text{PPL}_{\text{mask}} \approx \text{PPL}$. The practically important question is what happens when the rule is attached to a model that did not grow up under it — the subject of the next section.

## 4 Retrofitting Pretrained Models

### 4.1 Setup and controls

The retrofit experiments start from checkpoints pretrained for 3,000 steps without the rule (the §2 baselines). All fine-tunes run 600 steps — 20% of the pretraining budget — at reduced learning rate. Because a fine-tuned model has seen 600 more optimization steps than its parent, we compare against an *equal-step control*: the same checkpoint continued for 600 steps without the rule ("continued baseline"), whose PPL improves from 4.455 to 4.397. All costs below are quoted against this control, not the parent; quoting against the parent would flatter every fine-tuned number by about 1.3%.

### 4.2 Zero-shot attachment: the unadapted cost

Attaching the rule to the pretrained model with no training at all yields the guaranteed working set, and in our testbed it carried a substantial quality cost: $\text{PPL}_{\text{mask}} = 5.80 \pm 0.40$ at $m = 4$, +32% over the continued baseline, with the largest seed spread of any condition in this paper. The spread itself is informative: which roster the prefix happens to lock in matters greatly for a model whose experts were never shaped to work as a per-sequence team. We take this zero-shot number as the reference point for the training interventions below.

### 4.3 Fine-tuning under the soft rule helps; the barrier saturates

Fine-tuning with the rule active (soft phase only, $\delta = 0.3$) cuts the pinned cost to +5.2%. Under this configuration the soft-rule working set remains near 5.9 rather than $m = 4$: the pretrained router's score margins exceed the $2\delta$ barrier often enough to leak past it — unlike the from-scratch model of §3.4, which settles at the budget under the same $\delta$. Raising the barrier closes the gap: at $\delta = 0.6$ the working set lands at 4.11 and the pinned cost falls to +2.2%; $\delta = 1.0$ reproduces $\delta = 0.6$ to three decimals in both metrics. We read the $\delta = 0.6/1.0$ agreement as saturation — beyond the router's score range, a higher barrier changes nothing — which in practice removes $\delta$ from the tuning burden for retrofits (any sufficiently large value behaves alike), though we have verified this only on our testbed. We also observed the complementary regime: with *no* fine-tuning, a larger barrier hurts (pinned cost at $\delta = 0.6$ exceeds $\delta = 0.3$ by a wide margin), consistent with early roster lock-in on an unadapted router. $\delta$ thus plays different roles before and after adaptation, a distinction that mattered in practice and that we flag for anyone applying the rule zero-shot.

### 4.4 Fine-tuning under the deployed rule

The deployment condition is pinning (§3.2), so the natural train–test alignment is to fine-tune with pinning active (same prefix fraction; $\delta = 0.3$ governs the unpinned prefix) — the model

learns, for 600 steps, to complete sequences using only the roster its own prefix selected. Among the interventions we compared, this pinned fine-tune gave the lowest pinned cost: $\text{PPL}_{\text{mask}} = 4.437 \pm 0.011$, +0.9% over the continued baseline, and numerically below the model trained under the rule from scratch ($4.470 \pm 0.005$). We do not claim superiority over from-scratch training — the fine-tuned model has 3,600 total steps to the from-scratch model's 3,000, and the statement our data supports is parity. The seed spread also collapses ($\pm 0.011$ vs. $\pm 0.40$ zero-shot): adaptation removes the roster-lottery variance of §4.2. Overall, in this comparison, 600 steps recovered 97% of the gap between zero-shot attachment and the continued baseline.

### 4.5 What fine-tuning changed — and did not

Evaluated with the rule off, every fine-tuned model routes almost exactly like its parent ($\text{WS}_0$ within 0.1 of 7.8; $\leq 6\%$ of the working-set reduction retained). Fine-tuning did not teach the router sequence-locality in any form our measurements detect. Combined with §2, this supports a hypothesis — not a demonstrated impossibility — that durable router-level locality was not available at this fine-tuning length. What the measurements do jointly suggest is a different locus of change: with unconstrained routing unchanged and pinned cost collapsed, a natural hypothesis is that fine-tuning adapted the *experts* to the traffic the rule creates — each expert improving at serving whole sequences of the kinds that concentrate on it. We offer this as a hypothesis consistent with the measurements; distinguishing expert adaptation from subtler router changes would require intervention experiments we have not run. Either way, the practical conclusion stands: the weights need only cooperate with the rule, not carry it, and 600 steps of cooperation sufficed on our testbed.

## 5 Budget Curves, Scale, and the Erosion of Serveability

### 5.1 Cost versus budget ($N = 8$)

Applying the pinned fine-tune of §4.4 across budgets $m \in \{6, 4, 3, 2\}$ yields a cost curve of the kind a deployment could consult (Fig. 3, left): against the continued baseline, pinned cost rises smoothly as the budget tightens — +0.3% ($m = 6$), +0.9% ($m = 4$), +2.5% ($m = 3$), +6.5% ($m = 2$). The endpoint deserves a note. With top-2 routing, $m = 2$ is the smallest legal budget: the model completes every sequence with two experts per layer, fixed after the prefix — no within-sequence routing freedom except the gating within the pair itself. That this limit costs +6.5% on our task suggests that, here, most of the measurable value of expert diversity lies *across* sequences rather than within them. We state this as a property measured on our testbed; tasks with genuinely mixed sequences could price within-sequence freedom very differently, which is one reason §1 scopes our claims to single-domain sequences.

### 5.2 A second scale point ($N = 32$)

Repeating the pipeline at $N = 32$ (top-2, same data; 3 seeds) tests whether anything about the recipe is specific to 8 experts. The unconstrained baseline touches 27.7 of 32 experts per sequence per layer. Zero-shot attachment, tolerable at $N = 8$ (+32%, §4.2), is here of a different magnitude: pinning $m = 8$ without adaptation multiplies perplexity by roughly 9× over the continued baseline in our testbed. The same 600-step pinned fine-tune, however, brings the cost to +5.0% at $m = 8$ (a 75% reduction in resident routed-expert weights) and +12.7% at $m = 4$ (87.5%), with seed spread again collapsing ($\pm 0.005$ at $m = 8$, against $\pm 2.1$ zero-shot). Two comparisons are worth making explicit, with their limits. First, at matched 75% memory reduction, the cost at $N = 32$ (+5.0%) is

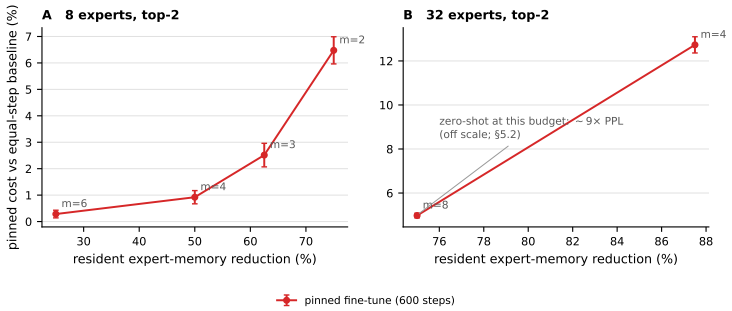



**Figure 3: Pinned-roster cost rises smoothly as the budget tightens.** Cost versus resident-memory reduction after the 600-step pinned fine-tune, quoted against equal-step continued baselines (3 seeds; error bars are std where visible). Two expert counts are shown; we do not fit or extrapolate a trend through them (§5.2).

below the cost at $N = 8$ (+6.5% at $m = 2$); this is consistent with the intuition that per-sequence expert demand grows sublinearly in $N$, leaving more slack at larger expert counts — but two scale points are two scale points, and we do not extrapolate a law from them. Second, the value of the fine-tune grew with scale in our experiments — from removing a +32% cost to removing a ∼9× one — which, if it persists at production scale, would likely make adaptation a requirement rather than an optimization; we flag this as the single most consequential unknown for §7.

### 5.3 Unconstrained training erodes serveability

The $N = 32$ runs additionally reveal a trend in the baseline itself. Tracking pinned-roster perplexity of the *unconstrained* baseline over its own training, serveability deteriorates monotonically while ordinary perplexity improves: $\text{PPL}_{\text{mask}}$ at $m = 8$ rises monotonically in every seed across 3,000 steps (e.g., 19 → 38 in one seed; three-seed mean at the end of training, 41.3), over the same span in which PPL falls 4.77 → 4.44. The trend continues past convergence: 600 further plain steps (the continued baseline itself) raise $\text{PPL}_{\text{mask}}$ from 41.3 to 43.6, again in every seed, while PPL improves by 0.06. At $N = 8$ on single-domain data we could not detect a corresponding change over 600 steps — the effect, if present, is below our resolution there. On mixed-domain data (§5.4), however, the same monotone erosion is visible at $N = 8$ across full training ($\text{PPL}_{\text{mask}}$ at $m$=4 rising 16.5 → 40.3, in both seeds), so detectability appears to depend on the difficulty of the demand as well as on expert count; on expert count itself we still have only two points. A mechanism consistent with these curves is that continued training keeps sharpening token-level routing preferences, which is exactly the axis a pinned roster cuts against; we offer this as a hypothesis, as we have not isolated it. The practical reading is more robust than the mechanism: on our testbed, a model's suitability for roster serving was not a fixed property — in our runs it declined monotonically as ordinary training proceeded. If this holds beyond our setting — an if we cannot yet discharge — it argues for exposing the serving rule during training, even briefly, rather than treating serveability as recoverable later at fixed cost.

### 5.4 A first mixed-sequence stressor

The pinned roster's most direct hazard is a sequence that changes topic midstream (§1, Scope). As a first, synthetic version of this stressor, we retrain the $N$=8 testbed on data in which *every* sequence switches to a different domain at a uniformly random point in $[0.3, 0.7]$ of its length (2 seeds; we report both). The hazard is real: zero-shot pinning at $m$=4, which cost +32% on single-

domain data (§4.2), costs roughly 7.6× perplexity here ($\text{PPL}_{\text{mask}} = 35.3 \pm 5.0$ against the mixed baseline's 4.67). A 600-step fine-tune under the deployed rule — trained with pinning plus periodic re-pinning every 64 tokens, the serving policy one would deploy for shifting demand — reduces this to +5.0% ($4.903 \pm 0.008$; both figures are quoted against the mixed parent checkpoint — we did not run an equal-step continued control for this stressor, and by the pattern of §4 such a control would make these percentages modestly larger). Two aspects of the adapted model were unexpected. First, at serving time the periodic re-pinning turned out to be unnecessary: single-pin and refresh serving give identical cost to three decimals, and at re-pin points the roster membership almost never changes ($\sim 10^{-5}$ changes per token). The measurements are consistent with the fine-tune teaching experts to complete the post-switch segment from the roster the first segment selected, though we have not isolated the mechanism. Second, the price of the switch itself is then modest: +5.0% on mixed sequences against +0.9% on single-domain ones at the same budget — about four points for one mid-sequence domain change. We note the limits plainly: our domains share part of their vocabulary, sequences switch exactly once, and we evaluated a single fine-tune configuration; real text can shift more sharply and more often, and this experiment does not predict that cost.

# 6 What Sequence-Local Demand Buys a Serving System

## 6.1 Method and accounting

We translate routing into serving terms by trace replay: we record every top-$k$ selection of the trained models on held-out sequences (256 sequences × 4 layers × 256 tokens) and replay the traces through a cache simulator. The simulator is deliberately simple, and we state its accounting in full so that the comparison can be audited. Both systems hold $C$ experts resident per layer — memory budgets are matched exactly. The unconstrained baseline is served by a fetch-on-miss LRU cache; we chose LRU because unconstrained token-level demand gives a cache little structure to exploit beyond recency, but we note that smarter predictive prefetchers exist — speculative prefetching from router logits, for example, is reported to outperform recency-based caching on Mixtral-class models [2] — and our baseline does not represent their ceiling. To bound what stronger prediction could achieve, we therefore also report an *oracle-overlap* variant of the baseline: prediction is perfect and every transfer overlaps compute fully, so the only stall is the per-token bandwidth residue $\max(0, \text{fetch time} - \text{compute time})$. This upper-bounds any predictive prefetcher at the stated bandwidth, since prediction hides transfer latency but does not reduce transfer volume. The roster model is served by the pinning schedule of §3.2: cold fetches during the prefix, a one-time prefetch of the roster at the pin point, and a fetch per barrier crossing thereafter. Transfers are charged at full cost with no compute overlap — an assumption that penalizes our method most, since its transfers are concentrated in one prefetch that a real system could overlap with prefill. Latency figures assume one system parameterization (expert weight 176 MB, host-to-device bandwidth 25 GB/s, 15 ms/token compute); transfer *counts* are properties of the traces alone and independent of these constants.

## 6.2 Results

At $N = 32$ with $C = 8$ (one quarter of expert weights resident), the two demand patterns produce different serving regimes (Table 3). The unconstrained model forces 1,108 fetches per sequence under LRU — routing touches ∼28 experts per layer and revisits them in no cache-friendly order — for a steady-state stall of 30.3 ms/token and effective throughput of 33% of the full-residence reference. The roster-served model performs 34.5 fetches per sequence: within 8% of the theoretical

| Serving configuration ($N$=32, $C$=8/layer) | Fetches/seq | GB/seq | Stall (tail) | Effective throughput |
|---|---|---|---|---|
| Full residence (reference) | 0 | 0 | 0 ms/tok | 100% |
| Unconstrained + LRU | 1,108 | 195.1 | 30.3 ms/tok | 33% |
| Unconstrained + oracle prefetch | 1,108 | 195.1 | 15.3 ms/tok | 50% |
| Roster-served (pin at $p$=0.2) | 34.5 | 6.1 | 0.01 ms/tok | 94% |

**Table 3:** Trace-replay serving comparison at matched memory (one quarter of expert weights resident). Fetch counts follow from the traces alone; GB, stall, and throughput additionally assume the system constants of §6 (176 MB/expert, 25 GB/s, 15 ms/token) and the conservative accounting stated there.

minimum of 32 (one roster prefetch per layer), with post-pin fetches at 0.0008/token. Its effective throughput, charging all warm-up and prefetch stalls, is 94% of full residence. The oracle-overlap baseline reaches 50%: unconstrained demand averages 4.3 fetches per token, whose transfer time exceeds per-token compute at this bandwidth, leaving a stall no prediction can remove — so the gap to roster serving is attributable to the demand itself, not to the baseline's cache policy, at a quality price (+5.0% perplexity) that the oracle does not pay. This comparison inherits the stated constants: above roughly 50 GB/s the oracle's residual stall vanishes and the throughput gap closes, while at slower interconnects it widens. The transfer-count ratio, 32×, is the number we regard as robust: it follows from the traces and the matched budget alone. The throughput ratio (2.8× here) inherits the system constants; under slower interconnects it widens (the baseline's stall grows while the roster model's remains near zero) and under faster ones it narrows toward parity, with the roster model approaching the full-residence ceiling from below in either case. Finally, the budget arithmetic composes with the quality results: taking §5's costs and this section's throughput together, one quarter of the expert memory purchased 94% of full-residence speed at +5.0% perplexity on our testbed — and at the limiting budget $m = k$, resident routed-expert weights equal their active-parameter share, the floor of §3.

### 6.3 What the simulation does not show

Three omissions bound these numbers. The traces come from teacher-forced evaluation; selections match what greedy decoding of the same tokens would produce, but free-running generation could drift to different expert demand, which we have not measured. The simulation covers a single stream; under batching, resident memory follows the union of per-sequence rosters — approaching $\min(B \cdot m, N)$ per layer for batch size $B$ when rosters do not overlap — and the savings depend on batch composition in ways we do not model. And the constants, while chosen to be representative of a commodity PCIe deployment, are inputs, not measurements: we present wall-clock validation on real hardware as necessary future work, not as something this section establishes.

## 7 Implications for Open-Weight MoEs: Arithmetic and Assumptions

This section extrapolates, and we ask the reader to treat it accordingly: nothing in it is established by our experiments, the numbers are order-of-magnitude estimates from public specifications rather than measurements, and readers interested only in established results can skip to §8 without loss. We include it because this arithmetic is what motivated the testbed study, and because stating the extrapolation's assumptions explicitly seemed more useful than leaving readers to reconstruct them.

### 7.1 The arithmetic, as an estimate

Under roster serving, resident weights would decompose as: always-active components (attention, embeddings, shared experts) plus a fraction $m/N$ of the routed-expert weights, per layer. As a worked example we take a recently released open-weight configuration reported as approximately 2.4T total parameters with ∼95B active per token, 512 routed experts with $k = 10$ plus one shared expert. We have not inspected the weights; from the public figures alone we estimate — and it is an estimate that a different parameter split would shift — roughly 2.3T parameters in routed experts and ∼50B in always-active components. On that split, full residency at 4-bit precision would be on the order of 1.2 TB, and roster serving at the same precision would come to very roughly 97 GB at $m = 32$, 61 GB at $m = 16$, and 48 GB at the floor $m = k = 10$, where resident routed-expert weights would match their active-parameter share. If the estimates and the assumptions below both held, the serving-memory question for such a model would shift from total parameters toward active parameters times $m/k$ — potentially a workstation-scale footprint. We phrase this in the conditional deliberately. Two costs deserve mention even within the optimistic case: non-resident experts still require storage with once-per-sequence read access, and at these sizes the roster prefetch itself would be tens of gigabytes, suggesting seconds of warm-up from commodity NVMe — amortized over long sequences, but not negligible, and we do not wish to minimize it.

### 7.2 The assumptions, stated

The estimate above rests on at least five assumptions that our experiments do not verify; there may be others we have not identified. (A1) *Budget ratios far beyond those tested.* $m = 16$ of 512 would be a 97% reduction; our tightest measured point is 87.5% (+12.7% perplexity, $N = 32$). Our two scale points are consistent with slack growing in $N$, but two points cannot license this extrapolation. (A2) *Fine-tuning transfers to heavily-trained models.* Our erosion observation (§5.3) suggests, if it extends beyond our setting, that a model trained on trillions of tokens may sit much farther from serveability than our 3,000-step baselines; whether a proportionally small fine-tune would still suffice — and what "small" would mean at that scale — is unknown to us, and we flagged it in §5.2 as the most consequential unknown. (A3) *Real text.* A first synthetic topic-shift test (§5.4) is encouraging — after adaptation, a mid-sequence domain switch cost +5.0% under a single pin, and periodic re-pinning proved unnecessary there — but its domains share vocabulary and switch exactly once. Real text can shift more sharply and more often, and that cost this paper cannot predict. (A4) *Architecture transfer.* Production MoEs differ from our testbed in fine-grained expert sizing, shared experts, larger $k$, and attention variants; each plausibly changes the demand statistics, in directions we have not measured. (A5) *Single-stream serving.* Batched serving would require the union of rosters (§6). Among these, (A2) and (A3) currently look to us like the load-bearing assumptions, though we hold that ranking loosely.

### 7.3 A possible path to validation

The assumptions appear individually testable, roughly in order of cost. Demand-trace analysis of existing open-weight MoEs on real text would require no training and could measure how sequence-local their routing already is and how candidate rosters behave across topic shifts (bearing on A3 and partially A4). A rule fine-tune of a small open MoE (1–7B class) could test whether the retrofit recipe survives contact with real data and give a first price for (A2) at accessible scale; models whose intermediate pretraining checkpoints are public would additionally allow the erosion observation of §5.3 to be tested directly on a real training trajectory rather than inferred from endpoints. In our view — and it is a view, not a result — the flagship-scale arithmetic of §7.1 would warrant direct

testing only after those two steps. We offer this ordering as the program our results suggest to us, not as work they accomplish.

## 8 Related Work

**MoE offloading systems.** A line of systems work serves MoE models whose experts exceed accelerator memory by caching and prefetching expert weights: LRU expert caches with speculative prefetching for Mixtral-class models [2, 4], and more generally parameter-offloading inference engines. These systems take the routing demand as given and engineer the serving side; we work on the other side of the same problem, asking whether the demand can be made easier to serve. The two are complementary rather than competing: in our traces, roster-served demand presents a prefetcher with a far simpler pattern — one prefetch per sequence, near-zero churn — than token-level demand does. Our LRU baseline in §6 is a representative of this family under matched memory, not a claim about its strongest member.

**Routing control through selection biases.** Our rule's form descends directly from auxiliary-loss-free load balancing [3], which steers top-$k$ selection with per-expert biases updated from load statistics, leaving gradients untouched; the design is deployed in production-scale models [5]. We transplant the same lever from the batch level to the sequence level and reverse its sign — concentrating within a sequence what balancing spreads across a batch — and add an explicit budget where balancing needs none. Earlier auxiliary-loss balancing and its interference costs [1] motivate the same selection-only discipline. To our knowledge, selection biases have not previously been used to shape *per-sequence* expert locality for serving; we would welcome pointers to precedents we have missed.

**Expert pruning, skipping, and specialization.** Static or task-conditioned expert pruning removes experts to shrink the model, and dynamic expert skipping drops low-gate experts at inference for speed [6]. Roster serving differs on both axes that matter here: nothing is removed — every expert continues to train and to serve the sequences that select it, and global load stays balanced in our runs — and the resident set is chosen per sequence at runtime rather than per model or per task offline. Work on expert specialization, including fine-grained expert designs [7], characterizes what routing concentrates on; our §2 analysis is narrower, asking only whether concentration itself can be made persistent by training, and finding, in our setting, that it could not.

**Budgeted-residency rules for other resources.** Structurally, our rule is an instance of a pattern that has proven productive for the KV cache: fix a memory budget, define a simple runtime policy for what stays resident (heavy-hitter tokens [8]; attention sinks plus a recency window [9]), and measure the quality retained. We borrow both the framing and the evaluation discipline — budget-matched comparisons, quality-versus-budget curves — and apply them to a resource where, to our knowledge, the budgeted-rule approach has not been standard: the expert weights themselves. The analogy also transfers a lesson: attention sinks proved cheaper to accommodate when exposed during pretraining [9], which parallels our §5.3 observation for serving rules.

# 9 Limitations and Conclusion

## 9.1 Limitations

We collect here the limits stated throughout, organized by which conclusions they bear on — and, for each, what a realized threat would do to that conclusion.

**Threats to the §2 analysis.** Our evidence that sequence-locality resists training comes from one synthetic task on which the language-modeling loss is nearly flat across routing patterns. In capacity-constrained regimes, where specialization pays in loss, both the reversion of §2.4 and the erosion of §5.3 could weaken or reverse; if they did, training-based approaches we set aside could become viable, and the case for a permanent rule would rest on its budget guarantee alone rather than on any necessity. A fourfold expert-width reduction did not change the reversion in our runs (§2.4), which narrows this caveat, but final loss remained routing-flat there, so it stands for tasks where specialization pays in *final* loss. We also tested a single distillation variant, and cannot rule out that an internalization scheme we did not try succeeds where ours failed.

**Threats to the retrofit results.** All fine-tunes are 600 steps from 3,000-step parents on the same distribution. Distribution shift between pretraining and fine-tuning — the standard situation in practice — is unexamined, and could raise the cost or the required length of the fine-tune by amounts we cannot estimate from our data.

**Threats to the serving numbers.** The simulator's omissions (§6) are not neutral simplifications; each could move the reported ratios. A strong predictive prefetcher could serve unconstrained demand better than our LRU baseline, narrowing the 32× transfer gap by an amount we cannot bound from our experiments. Free-running generation could produce demand our teacher-forced traces do not capture. Batched serving could reduce the memory savings toward the roster-union case. The throughput figures inherit assumed constants and could differ in either direction on measured hardware. Of our serving claims, we regard only the trace-level transfer counts under matched budgets as directly supported; the rest are conditional on the stated accounting.

**Threats to external validity.** Everything rests on single-domain synthetic sequences and two expert-count points; §7 enumerates what extrapolating beyond them assumes, and the failure of any of those assumptions would confine our conclusions to settings resembling the testbed. We note in particular a circularity risk in the testbed itself: the data were constructed so that each sequence has a well-defined home expert demand, which is precisely the precondition under which a pinned roster is most likely to succeed. Our results should therefore be read as characterizing the mechanism under favorable demand, not as evidence that such demand is typical of real text. A first mixed-sequence test (§5.4) partially addresses this — after adaptation, pinning survived a mid-sequence domain switch at +5.0% — but its domains share vocabulary and switch exactly once, so the favorable-demand qualification stands. We have tried to draw each conclusion no wider than the evidence beneath it; where we have failed, the failure is ours, not the data's.

## 9.2 Conclusion

We set out to ask whether MoE routing can be made sequence-local enough that serving memory follows a roster budget rather than the total expert count. On a controlled testbed, our attempts to train the property in did not persist: locality installed by selection pressure reverted under

continued training, and distilling it into the router destabilized routing in our experiments. We therefore imposed it instead as a routing rule, held active at training and serving alike. Under that rule, with a short fine-tune aligned to the deployed condition, pretrained testbed models served from a fraction of their expert memory at perplexity costs that rose with the tightness of the budget — from under 1% at half residency to several percent at the top-$k$ floor — and trace replay under matched memory showed the resulting demand requiring a small fraction of the transfers that unconstrained demand required of an LRU cache. Alongside, we observed that ordinary training steadily increased the cost of serving under a pinned roster. One reading consistent with these results, within the testbed's limits, is that per-sequence expert locality is more naturally handled as part of the routing procedure — specified, budgeted, and present during training — than as a behavior a model is asked to learn. Whether that reading holds for real text, production architectures, and heavily-trained models is not something this paper establishes; it is the question the paper's recipe and assumptions are laid out to let others test.

## Declarations

**Data and code availability.** The datasets generated and analyzed during the current study are synthetic and fully reproducible from the released code; the code, experiment configurations, and run outputs used in this study are available from the corresponding author on reasonable request, and will be posted in a public repository upon publication of the preprint.

**Competing interests.** The author declares no competing interests.

**Funding.** This work received no external funding.

**Ethics approval and consent to participate.** Not applicable. This study involves no human participants, human data, or animals.

**Author contributions.** C.S. conceived the study, designed and ran the experiments, analyzed the results, and wrote the manuscript.

**Use of large language models.** In accordance with preprint-platform policy on documenting LLM use, we note that a large language model (Claude, Anthropic) was used as an assistive tool throughout this work: implementing experiment code under the author's direction, aiding analysis of results, and drafting manuscript text. All experimental designs, decisions, and claims were directed and verified by the author, who reviewed all generated content and takes full accountability for the work. The LLM is not an author.

## References

[1] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 2022.

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

[3] Lean Wang, Huazuo Gao, Chenggang Zhao, Xu Sun, and Damai Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. In *ICLR*, 2025.

[4] Albert Q. Jiang et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[5] DeepSeek-AI. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[6] Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In *ACL*, 2024.

[7] Damai Dai et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*, 2024.

[8] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. $H_2o$: Heavy-hitter oracle for efficient generative inference of large language models. In *NeurIPS*, 2023.

[9] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *ICLR*, 2024.