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.

Which Expert Misses Are Worth Waiting For? Substitution and Caching for Mixture-of-Experts Inference on a Consumer GPU

ChaeWoo Son

soncwnetp@gmail.com

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

Research Article

Keywords: mixture of experts, expert offloading, expert caching, expert substitution, large language model inference, consumer GPU, gate weight, llama.cpp

Posted Date: October 6th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11268552/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.

Which Expert Misses Are Worth Waiting For? Substitution and Caching for Mixture-of-Experts Inference on a Consumer GPU

ChaeWoo Son *soncwnetp@gmail.com*

Abstract

Offloading lets mixture-of-experts (MoE) language models larger than GPU memory run on consumer GPUs. An expert missing from GPU memory can be fetched, computed elsewhere, or substituted with an expert already in GPU memory; substitution avoids the wait but changes the output. We ask which substitutions account for the loss in MoEs with hundreds of experts per layer, and whether the gate weight of the substituted experts can guide cache policies, mainly on Qwen3-Next-80B-A3B with a quarter of its experts in GPU memory and with quality measured mainly by teacher-forced perplexity. In an idealized emulation, fetching only the misses that were the router's first choice or had a gate weight of at least 0.15 raised perplexity by 0.35%, against 5.7% when every miss was substituted, although the substituted gate mass (the summed gate weight of substituted experts) fell by only about a quarter. In a llama.cpp-based engine, policies that always fetch the first choice tended to have higher perplexity on the same inputs when their mass was higher; in an exploratory comparison, however, mass did not track which inputs lost more. A rule that loads an expert in the background once its accumulated substituted mass reaches 0.5 lowered perplexity by 0.3–1.5%, most after topic changes. Two policies chosen from these results were 2.5–3.0× faster per token than fetching every miss at the same cache size, with 0.7–1.5% higher perplexity, on two hosts. On one RTX 3090 machine with about 16 GB of available RAM, the engine with the 0.15 rule decoded 2.8–3.0× as fast as stock llama.cpp configured for the same VRAM under teacher forcing (1.2–1.5× with ample RAM). A generation test on GSM8K and HumanEval, whose hypotheses and decision rules we fixed in advance, supported our predictions only in part. A second planned experiment, which held mass fixed and varied its composition, reached no judgment under its planned rule; in the same run and in exploratory conditions, substitutions concentrated on a few experts lost more than spread-out ones. Apart from these two planned experiments, the analyses are exploratory. We report every planned judgment, including those that failed, and two conclusions that reversed with more data, and we did not compare against prefetching or lower-precision methods.

1 Introduction

Mixture-of-experts (MoE) language models compute only a few experts per token but must keep all experts in memory. Qwen3-Next-80B-A3B, for example, uses about 3 billion parameters per token, yet even its Q4_K_M model file is 48.5 GB and does not fit on a 24 GB consumer GPU. Offloading keeps experts in CPU memory and moves them to the GPU on demand, so that such models can run on consumer GPUs (Eliseev & Mazur, 2023; Xue et al., 2025); recent systems also read experts from an SSD (Kim et al., 2026). Its cost comes from how it handles experts that are not in GPU memory, that is, cache-miss experts.

Four approaches to cache-miss experts are representative. Fetching missed experts on demand leaves the output unchanged but waits for a transfer on every miss; keeping recently used experts in a cache makes such waits less frequent (Eliseev & Mazur, 2023). CPU computation removes the transfer but waits for slower CPU arithmetic instead (Kamahori et al., 2025; Chen et al., 2025; llama.cpp, ggml-org, 2023). Prediction-based

prefetching hides the latency when the prediction is right, but spends bandwidth and leaves misses when it is wrong (Xue et al., 2025; Song et al., 2025; Wang et al., 2026; see Hoang et al., 2026 for a comparison of such policies). Fine-tuning the model to select the next block's experts one block ahead avoids mispredictions but changes the model (Hwang et al., 2024). Lower-precision copies of less important missed experts reduce the transfer at a small cost in accuracy (Tang et al., 2024). As the cache shrinks, misses become more frequent and all of these costs grow.

Other methods change the computation so that some missed experts are not fetched at all. They skip experts with small weights (Lu et al., 2024; Zhong et al., 2024), route toward experts already in memory (Kong et al., 2024; Skliar et al., 2025), or replace a missed expert with another one that is cached or predicted in advance (Du et al., 2024; Zhu et al., 2026; Wang et al., 2025; Madan et al., 2026). Reports differ on which experts can be changed. Replacing the top-ranked expert degraded performance severely and replacing the second noticeably, whereas replacing experts below the second rank cost little in models with 60–64 experts per layer (Skliar et al., 2025). In a benchmark with a small cache, however, dropping low-ranked missed experts still lowered accuracy, and results for score-based substitution differed between studies (Zhu et al., 2026; Hoang et al., 2026). A recent study on models with up to 128 experts per layer reports that mispredicted experts were mostly low-ranked with small routing weights (Madan et al., 2026). Most of these methods decide by rank or by a per-token score threshold, and most were evaluated with aggregate metrics on models with at most 64 routed experts per layer. In MoEs that select 6–10 of several hundred experts, experts other than the first choice can also carry large weights, so rank alone may not identify the experts to protect. To our knowledge, which substitutions account for most of the loss in such MoEs, whether one quantity can compare substitution policies on the same inputs, and when a cache should admit an expert that keeps being substituted have not been measured directly.

We examine which substitutions account for most of the quality loss, and whether this observation can inform cache policies. Our experiments use mainly Qwen3-Next-80B-A3B with a quarter of its experts in GPU memory, and combine three tools with different strengths and limitations. An emulation computes model outputs exactly but idealizes fetching. A router-logit simulator compares policies quickly without computing the model. A llama.cpp-based engine measures real fetches and speed. After choosing two policies from these results, we checked their accuracy on generation tasks in a test whose hypotheses and decision rules we fixed before running it. We also measured, in a second experiment planned in the same way (Experiment B, in the emulation), whether the composition of substitutions changes the loss at equal mass. We report every judgment of these two planned tests, including those that failed, and the conclusions that reversed when the sample grew. Our claims and their evidence are as follows.

Below, the 0.15 rule synchronously fetches a missed expert if it is the router's first choice or its normalized gate weight is at least 0.15 and substitutes the rest; the first-choice rule fetches only missed first choices; and mass is the summed normalized gate weight of the substituted experts, averaged over layer-tokens (§3.1). Each claim carries its grade of evidence; Appendix I gives the conditions, evidence and limits of each.

1. **In the emulation, most of the quality loss from substitution appears to come from a few substitutions of heavily weighted experts.** [exploratory · emulation · two models, weaker evidence from a third] (§5.1)
2. **Among policies that always synchronously fetch the first choice, compared on the same inputs, policies with more mass had larger perplexity increases. Mass did not, however, explain differences in loss between inputs.** [exploratory · engine · partly planned] (§5.2, §7)
3. **A rule that admits experts in the background according to their accumulated substituted mass (threshold 0.5) lowered perplexity, with the largest difference right after topic changes.** [exploratory · engine] (§5.3)
4. **On the two hosts we measured, two policies chosen from these results were 2.5–3.0× faster per token than fetching every miss, with 0.7–1.5% higher perplexity.** [chosen from the results · two hosts · one run per configuration] (§6.1)
5. **On one consumer GPU, the engine with the 0.15 rule and its default read path (page cache first) answered faster than stock llama.cpp configured to use the same VRAM, and the gap was largest at about 16 GB of RAM.** [one machine · 0.15 rule] (§6.2)
6. **A planned generation test only partly supported Claims 2 and 4.** [planned] (§7)

7. **A planned experiment that held mass fixed and varied only its composition reached no judgment under its planned rule. In the same run's auxiliary statistic and in exploratory conditions, a few heavy substitutions raised perplexity more than many light ones.** [planned (no judgment) · auxiliary statistic and exploratory · emulation] (§5.4)

This paper does not claim that gate weight itself causes the loss (we did not separate weight from rank, and Experiment B does not either); that equal mass implies equal loss; that mass predicts which inputs will lose quality; that emulation numbers carry over to the engine (the same 0.15 rule gave +0.35% and +1.21%); that the engine results hold for other models (the engine was tested on Qwen3-Next only); that our policies outperform prefetching or lower-precision methods (not compared); that the speed ratios hold on other machines; or that free-form generation quality is preserved in general (the generation test covers greedy generation on two tasks).

2 Related work

This section groups prior work on cache-miss experts into methods that keep the output unchanged (§2.1) and methods that change it (§2.2), and then relates our policies to work on cache replacement and admission (§2.3). We count only routed experts when we give the number of experts per layer.

2.1 Offloading without changing the output

Most systems keep experts in CPU memory (Eliseev & Mazur, 2023; Xue et al., 2025). Some read them from an SSD. FlashMoE reads experts from an SSD on edge devices and uses a replacement policy trained to imitate Belady's optimal policy from recency and frequency features (Kim et al., 2026). Concurrently, SGLang read experts from an NVMe SSD with direct I/O into a GPU cache on consumer hardware (WiCi AI Team & SGLang Team, 2026), and another study evaluates the operating system's page cache as the expert tier and favors kernel-managed eviction (Si et al., 2026). These systems compute every expert the router selects, so the output does not change.

They reduce the latency of misses in three main ways. First, they improve cache replacement. Besides least-recently-used caching (Eliseev & Mazur, 2023), frequency-based (Lin et al., 2025) and learned (Kim et al., 2026) replacement have been proposed, and an analysis of 20 MoEs reports that models differ in how consistently they route within short segments, and therefore in how well they suit offloading (Liang et al., 2026). Second, they predict upcoming experts and prefetch them (Xue et al., 2025; Song et al., 2025; Wang et al., 2026). SeqMoE reports average hit rates of 91.72% and 96.97% with 25% and 45% of the experts resident, on four models with 128–256 experts per layer (Wang et al., 2026), and SpecMD compares such prefetching and miss-handling policies under common conditions (Hoang et al., 2026). Third, they compute missed experts on the CPU (Kamahori et al., 2025; Chen et al., 2025; Yang et al., 2026; llama.cpp).

Other work changes the model itself. It fine-tunes a gate to select the next block's experts in advance (Hwang et al., 2024), fine-tunes the model to use fewer distinct experts per sequence (Raje et al., 2026), or post-trains the backbone and auxiliary routers to suit a cache (Wu et al., 2026). Our reference configuration, fetching every miss, keeps the output unchanged. We did not compare with prefetching, and our only comparison with CPU computation is stock llama.cpp, made with the 0.15 rule (§6.2).

2.2 Miss handling that changes the output

Methods that change the output can be grouped by what they change.

- **Skipping.** Lu et al. (2024) skip the second expert when its weight is small relative to the first, and AdapMoE uses one expert instead of two when a sensitivity estimate, which combines the token's top-1 gate score with a per-layer Fisher-information term, falls below a threshold (Zhong et al., 2024). MoBiLE halves the number of experts for tokens judged unimportant and falls back to all of them when the output is uncertain (Zhao et al., 2026).

- **Lower precision.** HOBBIT loads lower-precision copies of less important missed experts and skips the least important ones (Tang et al., 2024). SliceMoE splits experts into bit slices and computes non-critical (low-scoring) experts with the most significant bits only (Choi et al., 2026).
- **Re-routing.** SwapMoE keeps in memory a dynamically updated set of experts that are important over many tokens and routes only to it (Kong et al., 2024). Skliar et al. (2025) boost the router logits of cached experts to re-rank them, while always keeping the top one or two experts.
- **Substitution.** SMoE replaces experts whose scores lie near the selection boundary with cached experts of similar score (Zhu et al., 2026), and BuddyMoE replaces an expert whose prefetch failed with a "buddy" that is often activated together with it (Wang et al., 2025). SiDA-MoE (Du et al., 2024) and Madan et al. (2026) execute experts predicted in advance in place of the router's choices.

The evidence on which experts can be changed is mixed. Skliar et al. (2025) replaced the expert of each rank with a random expert. Replacing the top-ranked expert severely degraded performance and replacing the second degraded it noticeably, whereas replacing experts below the second rank cost little in models with 60–64 experts per layer. SMoE reports negligible accuracy changes for substitution thresholds below 0.35 (Zhu et al., 2026). With a cache of 5% of the model size, however, SpecMD found that dropping low-ranked missed experts still cost 5–30% of performance, and that its re-implementation of score-based substitution failed in every configuration (Hoang et al., 2026). In a study that executes predicted experts, the GSM8K accuracy of Qwen3-30B-A3B fell from 0.950 to 0.576 with router-based prediction and was 0.918 with an improved predictor; the same study reports that mispredicted experts were mostly low-ranked with small routing weights (Madan et al., 2026).

Most of these methods were evaluated on models with at most 64 experts per layer, the exceptions being SiDA-MoE and Madan et al. with up to 256 and 128, and most decide by rank or by a per-token score threshold. Our study differs in three ways. First, it measures substitution loss in three models with 128–512 experts per layer, and in the two with 128 and 256 experts it separates whether the loss comes mainly from substituting first choices, other heavy experts or light experts (§5.1). Second, it compares policies on the same inputs by the summed gate weight of substituted experts, the mass (§3, §5.2). Third, it uses this weight to decide cache admission (§5.3). Our substitute is the resident expert outside W with the largest logit, which is close to score-based substitution. We have not established why the reports on substitution differ; in our emulation, however, the loss depended strongly on which experts were substituted (§5.1).

2.3 Cache replacement and admission

Cache replacement has a long history. The optimal policy of Belady (1966) requires future requests and serves as a reference, and practical policies combine recency (LRU) and frequency (LFU). LRFU spans the two with a single decaying record of uses (Lee et al., 2001), and ARC adapts the sizes of two lists (Megiddo & Modha, 2003). Our frequency-based eviction counts the uses of router-selected experts as an exponential moving average. If LRFU's clock advances once per token rather than once per reference, this ranks experts as LRFU does with $\lambda = -\log_2(1 - 1/128)$.

What to admit into a cache can be decided separately from what to evict. TinyLFU admits a new item only when, judged by recent access frequencies, replacing the eviction candidate with it is expected to raise the hit ratio (Einziger et al., 2017). Our background admission rule instead accumulates the gate weight an expert loses each time it is substituted and admits the expert when the sum reaches a threshold. This has the form of a rent-or-buy (ski-rental) decision, which buys once the accumulated rent reaches the purchase price (Karlin et al., 1988). Because our rule measures the cost by gate weight and decays it with a half-life, the competitive guarantees of that problem do not carry over directly.

3 Hypothesis and scope

This section states the one hypothesis that our analyses rely on and the scope in which we apply it. We formed the hypothesis from exploratory results, and a later experiment whose hypotheses and decision rules were fixed in advance (§7) tested part of it. We also set the scope conditions after seeing the data. Whether the

hypothesis can ignore the composition of substitutions we tested in a second planned experiment, Experiment B (§5.4).

3.1 Setting and notation

At token t, the router of layer ℓ produces logits $z \in \mathbb{R}^E$ over E experts and selects the K experts with the largest logits. We call this set W and its expert with the largest logit the first choice e^1, dropping ℓ and t when they are clear. The normalized gate weight g_e is the softmax of the logits within W, so the weights sum to one (DeepSeek-V4-Flash instead divides its router scores by their sum within W). Qwen3-Next-80B-A3B has E = 512 and K = 10, plus one shared expert that is always computed.

Each layer keeps C experts in GPU memory, the resident set M. By default C = E/4, which is 128 experts per layer for Qwen3-Next. An expert in W but not in M is a cache-miss expert. A rule chooses the set F of misses to fetch synchronously and substitutes the rest, S = (W\M)\F:

- Fetching every miss: F = W\M. Nothing is substituted; this is the reference.
- The 0.15 rule: F holds the misses that are the first choice or have $g_e \geq \tau$, with the cutoff $\tau = 0.15$.
- The first-choice rule: F holds the first choice alone, when it is a miss.
- Substituting every miss: F = $\emptyset$. We used this only in the emulation.

Substitution is equivalent to selecting the K experts with the largest logits within M $\cup$ F. The slot of a substituted expert therefore goes to the resident expert outside W with the largest logit, and the gate weights are renormalized by a softmax over the new selection. The emulation and the engine substitute in this way, and the simulator applies the same rules to compute cache states and mass.

The substituted gate mass of a layer-token is the summed original weight of its substituted experts, $m = \sum_{e \in S} g_e$. The mass of a policy, $\bar{m}$, is the mean of m over all scored layer-tokens, counting layer-tokens without substitution as zero.

Two further components change the cache. Frequency-based eviction evicts the resident expert with the lowest frequency score, an exponential moving average with weight 1/128 per token that counts only experts selected by the router; with this component, the initial resident set after the prompt is also filled by frequency score. The background admission rule adds g_e to an expert's account each time the expert is substituted, and the account decays with a half-life of 512 tokens. When the account reaches a threshold β, the expert is read in the background, replaces the resident expert with the lowest frequency score, and its account is reset; at most eight such admissions can be pending per layer, and four loader threads shared by all layers serve them. In the engine, the resident set changes only through synchronous fetches and this rule. The emulation instead assumes that every missed expert selected by the router is resident from the next token. In this paper, a policy is a combination of a rule, these two components and the cache size C.

We measure loss as the relative increase in perplexity over fetching every miss on the same tokens, $\Delta\text{ppl} = \text{ppl}_\pi/\text{ppl}_{\text{exact}} - 1$. For generation tasks we use accuracy differences and agreement with the reference configuration on the same problems. By the same inputs we mean the same token sequences under teacher forcing and the same problems in generation.

3.2 Hypothesis

Mass hypothesis. Among policies that always fetch the first choice, compared on the same inputs, policies with more mass lose more quality to substitution.

The hypothesis concerns order. It does not state that mass causes the loss or that the loss is proportional to mass. The slopes we observed, 0.25–0.26 perplexity points per 0.01 of mass (§5.2), summarize the measured conditions and are not part of the hypothesis.

Why mass. At one layer-token, let $y = \sum_{e \in W} g_e f_e(x)$ be the summed output of the routed experts, and $y' = \sum_{e \in W'} g'_e f_e(x)$ the sum after substitution, where W′ is W with S replaced by the substitutes S′ and g′ is renormalized within W′. For softmax gates, as in Qwen3-Next and gpt-oss, the following holds. The

substitutes have smaller logits than every expert in W, so the weights of the retained experts do not decrease. Because the weights of both selections sum to one, the increases in the retained weights and the weights of the substitutes add up to exactly m. If every expert output satisfies $\|f_e(x)\| \le F$, then

$$\|y - y'\| \le \sum_{e \in S} g_e \|f_e(x)\| + \sum_{e \in S'} g'_e \|f_e(x)\| + \sum_{e \in W \setminus S} (g'_e - g_e) \|f_e(x)\| \le 2F\, m.$$

Mass thus bounds how much the output of one layer can change.

The bound is only a starting point. First, as an upper bound, it may be far from the actual change, which depends on the directions of the expert outputs and spreads nonlinearly through later layers and tokens. Second, the bound depends on m alone and cannot tell which experts were substituted. If the changes from individual substitutions pointed in unrelated directions, for example, their combined size would grow more like $(\sum_{e \in S} g_e^2)^{1/2}$ than like m, and a few heavy substitutions would change the output more than many light ones of the same total mass. If each substitution instead carried a cost unrelated to its weight, the loss from many light substitutions could be larger. In Experiment B (§5.4), a few heavy substitutions lost more than many light ones at equal mass, although its planned decision rule reached no judgment.

3.3 Scope conditions

Two conditions limit the hypothesis. We set both after seeing the data, and we added the second after a prediction fixed in advance failed.

Condition 1: the first choice is always fetched. When policies that also substitute first choices are included, mass does not follow the order of the losses. In the Qwen3-Next emulation, the 0.15 rule with a 12.5% cache had more mass than substituting every miss with a 25% cache (0.132 vs. 0.076) but a smaller perplexity increase (1.62% vs. 5.67%). Even when the order holds, the losses can differ far more than the masses. With a 25% cache, substituting every miss and the 0.15 rule differed by 26% in mass (0.076 vs. 0.056) but 16-fold in perplexity increase (5.67% vs. 0.35%). In DeepSeek-V4-Flash, substituting every miss with a 25% cache and the 0.15 rule with a 12.5% cache differed by 6% in mass (0.132 vs. 0.124) but about 18-fold in perplexity increase (4.97% vs. 0.28%). Substituting every miss, however, also substitutes misses with weights of 0.15 or more, so these comparisons include the share of other heavy substitutions as well as that of first choices.

Condition 2: policies are compared on the same inputs. Mass followed the loss order of policies on the same inputs, but not the differences in loss between inputs. In the engine, the correlation between the mean mass and the mean loss of input groups was -0.21 with 4 groups and -0.32 with 8 groups, and within each policy it ranged from -0.44 to -0.02. In the planned generation test, the prediction that problems with more mass would change their answers more often (H3) failed (both tasks pooled: 0.165 log-odds per 0.1 of mass, $[-0.048, +0.393]$). On GSM8K, a problem's mass was negatively correlated with its generation length (-0.49 to -0.71), so properties of the input such as length appear to be mixed with mass. We therefore use mass only to compare policies on the same inputs, not to predict which inputs will lose quality.

In addition, we have checked the hypothesis only under the conditions we measured: one model in the engine (Qwen3-Next), mostly a 25% cache, and quality measured by teacher-forced perplexity and by greedy generation on two tasks.

3.4 Predictions and their status

Table 1 lists the predictions that follow from the hypothesis and its conditions, and one prediction outside the scope. "Fixed in advance" means that the prediction and its decision rule were fixed before we saw the data.

Table 1: Predictions that follow from the hypothesis and its conditions, one prediction outside the scope, and their status.

	Prediction	Data	Fixed in advance	Result
P1	On the same inputs, policies rank the same by mass and by perplexity	Engine, single topic, 24 sessions, 7 policies (R18)	No; the hypothesis was formed from these data	Pearson 0.92, Spearman 0.86; three pairs out of order (below)
P1′	The same, on new sequences	Engine, topic change, 24 sequences, 6 policies (R19)	No; the sequences were new, but the analysis was not fixed in advance	Pearson 0.91, Spearman 1.00
P2	In the order of mass, agreement with the reference falls (H2) and accuracy drops (H6)	Generation, GSM8K, 500 problems, 5 configurations (R20)	Yes	Held. Agreement Spearman -0.975 (token sequences) and -0.9 (final answers; threshold -0.9); accuracy slope -0.41 points per 0.01
P2′	The same	Generation, HumanEval, 164 problems	Yes	Agreement failed (token identity -0.60, final outcome -0.67); the accuracy slope was negative (-0.09 points per 0.01) but $+0.16$ without H
P3	(Outside the scope) Problems with more mass change their answers more often (H3)	Generation, both tasks pooled	Yes	Failed: 0.165 [-0.048, $+0.393$]
P4	At equal mass, the loss does not depend on composition (a few heavy vs. many light substitutions)	Emulation, 48 sessions (Experiment B)	Yes (§5.4)	No judgment (Δppl of the light arm indistinguishable from zero). The heavy arm lost more in a difference not used for the judgment ($+0.41$ [$+0.13$, $+0.70$]); at an exploratory target of 0.15, $+4.35\%$ vs. $+0.49\%$

In the same test, the order of mass itself (H1) held on both tasks, but the masses were about twice the predicted values. H4 and H5 concern the accuracy of the chosen policies and are reported in §7; the first part of H4 (point estimates of four configurations within -1.5 to $+0.5$ points) failed on both tasks.

Exceptions within the scope. Within the scope, the order of mass and the order of loss sometimes disagreed. All of these comparisons are exploratory, and most individual differences lie within their intervals.

- Engine, 24 sessions (R18): the 0.15 rule with frequency-based eviction and background admission at a threshold of 1 had 14% less mass than the same policy without admission, but the same perplexity increase ($+0.95\%$ vs. $+0.94\%$). In the same run, the speed-first policy, from the first-choice-rule family, had less mass than these two policies (0.078 vs. 0.080 and 0.093) but a slightly larger perplexity increase ($+1.00\%$ vs. $+0.95\%$ and $+0.94\%$).
- Engine, 6 sessions, 12.5% cache (R17): the first-choice rule with frequency-based eviction and a threshold of 1 had less mass than the 0.15 rule (0.184 vs. 0.198) but a larger perplexity increase ($+3.61\%$ vs. $+2.85\%$; difference $+0.73\%$ [-0.13, $+1.64$]).
- DeepSeek-V4-Flash emulation: the first-choice rule with a 25% cache had less mass than the 0.15 rule with a 12.5% cache (0.1095 vs. 0.1238) but a larger perplexity increase (1.29% vs. 0.28%).

Apart from the first, these exceptions point the same way: with similar or smaller mass, policies of the first-choice-rule family, which also substitute misses with weights of 0.15 or more, lost more than policies of the 0.15-rule family. Such exceptions would arise if the composition of the substituted weight affected the loss. The two families also change the cache differently, however, so these comparisons cannot isolate an effect of composition. Experiment B (§5.4) varied only the substitutions, without a cache, to measure this effect separately.

4 Experimental setup

This section describes the models, data, tools, measures and statistics used in Sections 5 and 6. The setup of the generation test is given in Section 7, implementation and validation details are in Appendices A (engine) and B (emulation and simulator), and data and runs are in Appendix C.

4.1 Models and data

We use three models. Our main model, Qwen3-Next-80B-A3B (Instruct; Qwen Team, 2025; Yang et al., 2025), selects 10 of 512 routed experts in each of its 48 layers and adds one shared expert. We use its BF16 weights in the emulation and the Q4_K_M GGUF file published by unsloth (48.5 GB; Unsloth AI, 2025) in the engine. The other two models appear only in the emulation. DeepSeek-V4-Flash (DeepSeek-AI, 2026) selects 6 of 256 experts in each of 43 layers, the first three of which are hash-routed; we dequantize its FP8 weights and FP4 experts to bf16 and emulate the FP8 activation quantization of its official inference code block by block. gpt-oss-120b (OpenAI, 2025) selects 4 of 128 experts per layer, and we dequantize its MXFP4 experts to bf16 (numerical details in Appendix B.1).

The data are sessions built from public datasets in six domains: Korean and English Wikipedia articles (Wikimedia Foundation, 2023), Python source files (CodeParrot, 2021), English chats (UltraChat 200k; Ding et al., 2023), Korean chats (koVast; maywell, 2024) and math solutions (NuminaMath-CoT; LI et al., 2024). Each session is one context of 2,048 tokens: the beginning of a single long document or conversation, or, when such items ran short, shorter items packed back to back; math solutions are always packed (Appendix C.1). With a fixed seed we built up to 32 sessions per domain (18 for Korean chats). In each session the first 512 tokens are a prompt computed with the original routing, and quality is measured on the remaining tokens (1,535 next-token predictions). - Emulation: 48 sessions, 8 per domain (about 74k scored tokens). For DeepSeek-V4-Flash and gpt-oss-120b, the sessions were rebuilt by the same procedure with each model's tokenizer and chat format. - Engine, single topic: 24 sessions, 4 per domain, drawn by the same procedure. - Engine, topic change: 24 sequences that concatenate these sessions. Each sequence has a 512-token prompt and 4,096 scored tokens, and its domain changes at scored tokens 1,536 and 3,584. Each session appears once in each of the three measured positions, and the three parts of a sequence always come from different domains.

4.2 Three tools

Emulation. A PyTorch implementation computes the model layer by layer: it loads each layer's weights once and computes that layer for all sessions and all policies. Because it computes the outputs of the original routing and of every policy exactly, losses can be compared on the same tokens. The cache exists only as bookkeeping. The resident set starts from the prompt's routing (most recently used experts first). Every missed expert selected by the router is assumed resident from the next token, with least-recently-used eviction that never evicts an expert used at that token. Fetch time is not modeled. On small random models, the layer-wise forward pass of our routing-trace tool, whose token mixers, norms and weight loading the emulation reuses for Qwen3-Next, matched transformers (Wolf et al., 2020) (the outcome of a test of the emulation itself was not recorded), and so did the emulation for gpt-oss-120b. For DeepSeek-V4-Flash we compared modules on a small random model with the official inference code, its GPU kernels replaced by torch equivalents: the routing, compressed KV cache and expert outputs matched bit for bit, leaving differences of about one bf16 ulp from summation order (Appendix B.1).

Router-logit simulator. A C++ program replays router logits recorded with stock llama.cpp under the original routing and applies a policy to them, counting hits, mass, synchronous fetches and background reads, from which a model estimates the time per token. It does not compute the model, so it can compare dozens of policies quickly, but it misses the effect of substitution on the routing of later layers. Its predicted changes in mass and time between policies were within 0.1–3.6 and 0–3.3 percentage points of the engine's measurements (25% and 12.5% caches; the simulator on 24 sessions and the engine on 6 of them; Appendix B.2, Table F10).

Engine. Our engine adds an expert cache to llama.cpp (ggml-org, 2023). Each layer keeps slots in GPU memory (128 per layer with a 25% cache, 10.9 GiB in total), and a missed expert is read from the model file with O_DIRECT (§6.2 and §7 also use a path that copies an expert from RAM when it is in the page cache). Synchronous fetching, substitution, frequency-based eviction and background admission follow the definitions in §3.1. Without frequency-based eviction, the least recently used expert is evicted (substitutes actually used count as uses), and the initial resident set after the prompt is filled with the experts most recently used in the prompt; with it, the initial resident set is also filled by frequency score. Quality is measured by teacher forcing: the engine decodes the data tokens one at a time. For each session it saves the state after the prompt, and every configuration processes the same tokens from that state, so each policy's perplexity can be paired token by token with fetching every miss in the same run. On a small Qwen3-Next model (32 experts, top-4), we compared the engine's decision trace with an independent Python implementation: all decisions at 25,317 layer-tokens with misses and all 623 background admissions matched. The teacher-forced measurements ran on RTX 3090 (24 GB) hosts of the RunPod community cloud, except the R7 accuracy check cited in §6.2, which ran on an A40 in the secure cloud. Quality was the same across hosts (§5.2), whereas speed differed, so we report speed together with host specifications in §6. Implementation and host details are in Appendix A and Table C3.

4.3 Measures and statistics

Quality loss is Δppl from §3.1, the relative increase in perplexity over fetching every miss on the same tokens. In the emulation the reference is the original model, and we also report the KL divergence of the next-token distribution from the original model's and the argmax agreement with it.

All intervals for the engine are 95% percentile intervals from 2,000 paired bootstrap resamples on the same tokens. Block intervals resample 128-token blocks, each within one session or sequence, from the pooled set of a run's blocks (Appendix D.2), and sequence intervals resample whole sequences; the latter also capture differences between sequences and are therefore more conservative. Because the 24 sequences share sessions, however, even sequence intervals may be somewhat narrow. For topic changes we also run a two-sided sign test on the number of sequences in which one configuration has lower perplexity.

We examine the relation between mass and loss at three levels. At the policy level we fit a line to the policies' (mass, Δppl) pairs, give a 95% interval for the Pearson correlation by the Fisher transform, and report leave-one-out slopes and the Spearman correlation. At the group level we use the chunks into which the runs were split: 4 chunks of 6 sessions for the single topic and 8 chunks of 3 sequences for topic changes. The within-group (fixed-effects) slope is fitted after subtracting each group's means, with an interval from 2,000 bootstrap resamples of groups (seed 14). Between inputs, we correlate group means. With only 4 and 8 groups these analyses are coarse, and all of them were done after we saw the results.

4.4 Evidence grades and records

Every result carries an evidence grade. [Planned] means that hypotheses, predictions and decision rules were fixed before the run (the generation test of §7 and Experiment B in §5.4). [Exploratory] marks an analysis not fixed in advance. [Chosen from the results] marks values measured on the data used to choose them. Experiments proceeded as numbered runs of a research log; "Rn" in the text is the n-th run, listed in Appendix C (Table C2). The author used an AI system (Claude) as a research assistant: it wrote and maintained the code, ran the experiments and wrote first drafts that the author and Claude revised together; the section "Use of an AI system" after Section 8 describes its use. We report conclusions that reversed as the sample grew.

5 Which substitutions account for the loss

This section asks four questions in turn. In the emulation, which substitutions account for most of the loss (§5.1)? In the engine, does mass follow the loss order of policies that always fetch the first choice, compared on the same inputs (§5.2)? Does admitting experts that keep being substituted reduce the loss (§5.3)?

Does the composition of substitutions change the loss at equal mass (§5.4)? The first three analyses are exploratory; the fourth is a planned experiment.

5.1 Emulation: substituting first choices and heavy experts [exploratory]

Table 2: Emulation with a 25% cache and LRU replacement. Δppl, KL and argmax agreement are measured against the original model. The substitution rate is the share of expert calls that were substituted.

Model	Policy	Δppl	KL	Argmax agreement	Substitution rate	Mass
Qwen3-Next-80B-A3B	substitute every miss	+5.67%	0.085	92.3%	8.5%	0.076
	0.15 rule	+0.35%	0.013	96.5%	7.5%	0.056
DeepSeek-V4-Flash	substitute every miss	+4.97%	0.101	91.3%	15.9%	0.1320
	first-choice rule	+1.29%	0.042	93.6%	14.6%	0.1095
	0.15 rule	+0.20%	0.021	95.3%	11.2%	0.0715
gpt-oss-120b (code)	substitute every miss	+9.7%	–	–	15.2%	–
	first-choice rule	+1.19%	0.0152	96.5%	12.6%	–
	0.15 rule	−0.03%	0.0031	98.5%	1.65%	–

In DeepSeek-V4-Flash, the three hash-routed layers fetch every miss under every policy and so have no substitutions; substitution rates and masses are averaged over the other 40 layers. gpt-oss-120b is trained for its chat format, and its perplexity on plain text is very high even without substitution, so we report its quality on the code domain only, although it was measured on all six. Its substitution rates are over all six domains, and its mass was not recorded.

Table 2 shows the emulation with a 25% cache. In Qwen3-Next, the 0.15 rule synchronously fetched only 0.8% of expert calls and still substituted 7.5% of them. It thus kept about nine-tenths of the substitutions made when every miss was substituted (8.5%), yet its perplexity increase was 0.35%, against 5.67% (KL 0.013 vs. 0.085), and its mass was 26% lower (0.056 vs. 0.076).

DeepSeek-V4-Flash also shows the first-choice rule. The policy that fetched only missed first choices had a perplexity increase of 1.29%, against 4.97% when every miss was substituted, and the policy that also fetched misses with weights of 0.15 or more had 0.20%, while the substitution rates of the three policies differed much less (15.9%, 14.6% and 11.2%). Most of the loss thus appears to come from substituting first choices and other heavy experts, and little from the light substitutions that make up most of the substitutions. The code domain of gpt-oss-120b shows the same order (+9.7%, +1.19%, −0.03%). In this model, which selects only four experts per token, the 0.15 rule substituted only 1.65% of calls, against 15.2% when every miss was substituted, so it offers weaker evidence that the loss comes from few substitutions.

A 12.5% cache gave the same picture. Substituting every miss raised perplexity by 18.8% in Qwen3-Next and 13.9% in DeepSeek-V4-Flash, and the 0.15 rule by 1.62% and 0.28%.

These results have three limits. First, the emulation assumes zero fetch latency; the same 0.15 rule raised perplexity by 1.21% on the engine's 24 sessions (Q4_K_M, real fetches; §5.2). Second, only DeepSeek-V4-Flash and gpt-oss-120b separate first choices from other heavy experts; we did not run the first-choice rule for Qwen3-Next in the emulation. Third, these comparisons change the weight and the rank of the substituted experts together. In Experiment B (§5.4), which varied only the composition at equal mass, heavy substitutions also lost more, but that experiment cannot separate weight from rank either.

5.2 Engine: mass and the loss order on the same inputs [exploratory]

Table 3: Engine with a 25% cache. Δppl is relative to fetching every miss in the same run; brackets give block intervals (for topic changes, block interval; sequence interval). "Eviction" is frequency-based eviction, which also fills the initial resident set by frequency score, and "admission" is background admission.

Policy	Single topic: mass	Single topic: Δppl (%)	Topic change: mass	Topic change: Δppl (%)
0.15 rule	0.0978	+1.21 [+0.73, +1.69]	0.1337	+1.77 [+1.40, +2.13; +1.40, +2.16]
first-choice rule	0.1101	+1.76 [+1.27, +2.33]	0.1600	+3.27 [+2.76, +3.79; +2.59, +3.93]
0.15 rule + eviction	0.0930	+0.94 [+0.59, +1.32]	0.1327	+1.67 [+1.32, +2.02; +1.24, +2.11]
0.15 rule + eviction + admission ($\beta = 1$)	0.0802	+0.95 [+0.60, +1.31]	–	–
quality-first: 0.15 rule + eviction + admission ($\beta = 0.5$)	0.0717	+0.67 [+0.31, +1.04]	0.0851	+0.97 [+0.67, +1.30; +0.53, +1.36]
first-choice rule + eviction	0.1037	+1.55 [+1.03, +2.11]	0.1587	+3.02 [+2.48, +3.55; +2.25, +3.78]
speed-first: first-choice rule + eviction + admission ($\beta = 0.5$)	0.0779	+1.00 [+0.54, +1.50]	0.0929	+1.46 [+1.06, +1.87; +0.97, +1.98]

Table 3 lists the policies we measured in the engine (Figure 1a, b), all of which fetch the first choice synchronously. Over the 24 single-topic sessions, the correlation between mass and Δppl across the seven policies was 0.92 (95% interval 0.53–0.99), and each 0.01 of mass came with a 0.25-point larger Δppl; refitting without each policy in turn gave slopes of 0.21–0.27. Over the 24 topic-change sequences, the six policies gave a correlation of 0.91 (0.37–0.99) and a slope of 0.26 points (0.22–0.30 when leaving one out). The Spearman correlations were 0.86 and 1.00.

The relation also held within groups of inputs. The within-group slopes were 0.23 points [0.20, 0.31] with 4 groups and 0.26 points [0.20, 0.32] with 8 groups, and in all 12 groups the Spearman correlation between the policies' mass ranks and Δppl ranks was 0.79–1.00.

Between inputs, no such relation appeared. The correlation between the groups' mean mass and mean Δppl was −0.21 (4 groups) and −0.32 (8 groups), and within each policy it ranged from −0.44 to −0.02. Inputs with more mass did not lose more; this is Condition 2 in §3.3.

Measurement noise was small. The same session gave the same Δppl and argmax agreement to four decimal places on different hosts (the six sessions shared by R17 and R18), and rerunning a configuration changed perplexity by less than 0.04%. The conclusion was sensitive to sample size, however. With the 6 sessions measured first (R17, one per domain) the correlation was 0.77, and with the 0.15 rule, policies with background admission had less mass but not lower Δppl. We then concluded that mass was a poor proxy for quality; 24 sessions reversed this conclusion.

Within the scope, some pairs of policies were out of order (§3.4), and at similar mass the first-choice-rule family lost more. In the planned generation test, GSM8K accuracy differences followed the order of mass, but agreement with the reference configuration on HumanEval did not (§7).

5.3 Background admission [exploratory]

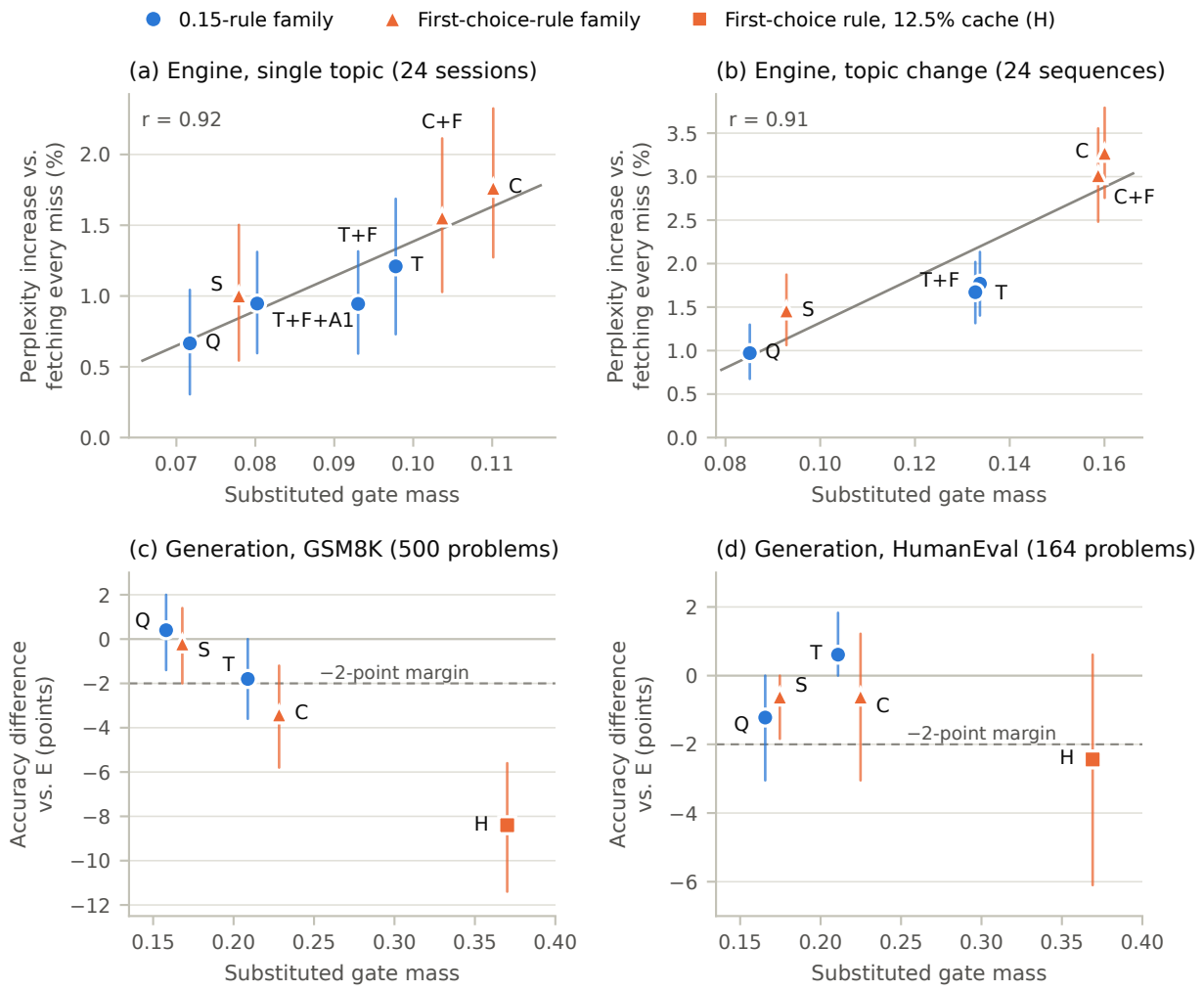


Figure 1: Substituted gate mass and quality loss across policies. Each point is one policy, with its 95% interval. (a, b) Engine with a 25% cache under teacher forcing [exploratory · engine]: perplexity increase over fetching every miss (E) on the same tokens, with block intervals (Table 3), on 24 single-topic sessions (R18) and 24 topic-change sequences (R19). The gray line is a least-squares fit over the policies (r = 0.92 with 7 policies and 0.91 with 6), computed after the results were known; panel (a) is part of the data from which we formed the mass hypothesis (Table 1). (c, d) Greedy-generation test [planned]: accuracy difference from E in points, with problem-level paired-bootstrap intervals (Table 9). The dashed line is the −2-point non-inferiority margin. Mass in (c, d) was measured during generation and is not comparable in absolute value with (a, b). T: 0.15 rule; C: first-choice rule; F: frequency-based eviction; A: background admission with threshold 0.5 (A1: threshold 1, shown only in (a)); Q = T+F+A (quality-first); S = C+F+A (speed-first); H: first-choice rule with a 12.5% cache. Mass is the summed original gate weight of the substituted experts of a layer-token, averaged over all decode layer-tokens (§3.1).

	0.15-rule family	First-choice-rule family

Table 4: Change in perplexity (%) of a policy with background admission ($\beta = 0.5$) relative to the same policy without it, on the same tokens (both with frequency-based eviction). Brackets give block intervals. For topic changes they give block; sequence intervals, and the rows for whole sequences and windows also give the number of sequences (of 24) in which the policy with the rule had lower perplexity.

	0.15-rule family	First-choice-rule family
Single topic, 24 sessions (R18)	−0.28 [−0.53, −0.04]	−0.54 [−0.84, −0.26]
Topic change, 24 sequences (R19)	−0.69 [−0.93, −0.43; −1.18, −0.24], 20/24	−1.51 [−1.85, −1.18; −2.07, −0.95], 20/24
— 512 tokens after each change	−1.34 [−2.00, −0.59; −2.32, −0.36], 17/24	−3.53 [−4.55, −2.46; −5.01, −2.07], 20/24
— remaining tokens	−0.47 [−0.68, −0.23; −0.89, −0.09]	−0.83 [−1.07, −0.58; −1.21, −0.40]
Change in time per token (single topic / topic change)	+0.4% [−0.6, +1.2] / +0.8% [+0.2, +1.5]	−0.4% [−1.1, +0.4] / −0.6% [−1.4, +0.5]

We chose the half-life of 512 tokens and thresholds of 0.5 and 1 from simulator results (R16; Appendix B.2 describes the choice). In the engine we measured thresholds of 0.25, 0.5 and 1 on 6 sessions (R17); on 24 sessions we measured 0.5 and 1 with the 0.15 rule, and only 0.5 with the first-choice rule and for topic changes (R18 and R19). Table 4 compares policies with and without the rule on the same tokens. With $\beta = 0.5$, the policies with the rule had 0.28% (0.15 rule) and 0.54% (first-choice rule) lower perplexity within a topic. Across topic changes the differences grew to 0.69% and 1.51%; in both cases the policy with the rule was lower in 20 of 24 sequences (sign test p = 0.002), and the sequence intervals excluded zero.

The differences concentrated right after topic changes (Figure 2; Table F4). The 512-token windows after the two changes hold 25% of the scored tokens, yet about half of the loss of the engine with the 0.15 rule alone came from them (windows +3.58%, remaining tokens +1.17%). The rule had its largest effect in these windows (−1.34% and −3.53%). Across topic changes its effect was more than twice as large for the first-choice-rule family, apparently because experts of a new topic enter the cache late when only first choices are fetched. Policies with the rule had 23–41% less mass, read 3–5 more experts per token in the background, and differed in time per token by −0.6% to +0.8%.

These results also have limits. A threshold of 1, tried with the 0.15 rule, made no difference within a topic (+0.00% [−0.19, +0.19]); it was not measured on the 24 topic-change sequences (R19). With the 6 sessions measured first (R17), the single-topic effect of a threshold of 0.5 with the 0.15 rule was not significant (+0.30% [−0.15, +0.76]). In our topic-change sequences the topic changes in the middle of the text decoded under teacher forcing rather than with a new prompt, which is likely harsher than a real conversation, in which a new topic usually arrives with a user prompt (we did not measure this). The sequences were also built from the same 24 sessions as the single-topic runs, so the two data sets are not fully independent.

5.4 Equal mass, different composition: Experiment B [planned · partly exploratory]

The mass hypothesis looks only at the summed weight of the substituted experts, not at how that sum is split among experts. As §3.2 noted, the output may change differently at equal mass when the composition differs, and the exceptions in §3.4 may have arisen from such differences. Experiment B measures this by holding mass fixed and varying only the composition. We fixed its design and decision rules before the run (Appendix G).

Design. We used the Qwen3-Next emulation of §5.1 without a cache, on 48 sessions (8 per domain) that avoid the sessions of the earlier emulation runs and, we believe, those of the engine runs (we did not check the latter on token files; Appendix G.2). Each session has a 512-token prompt with the original routing and 1,535 scored tokens (73,680 in total). At every scored layer-token, each arm takes W and g from its own logits and considers the subsets A of the nine experts other than the first choice, with mass $s(A) = \sum_{e \in A} g_e$ and concentration $c(A) = \sum_{e \in A} g_e^2 / s(A)^2$ (1 for a single expert, $1/n$ for n experts of equal

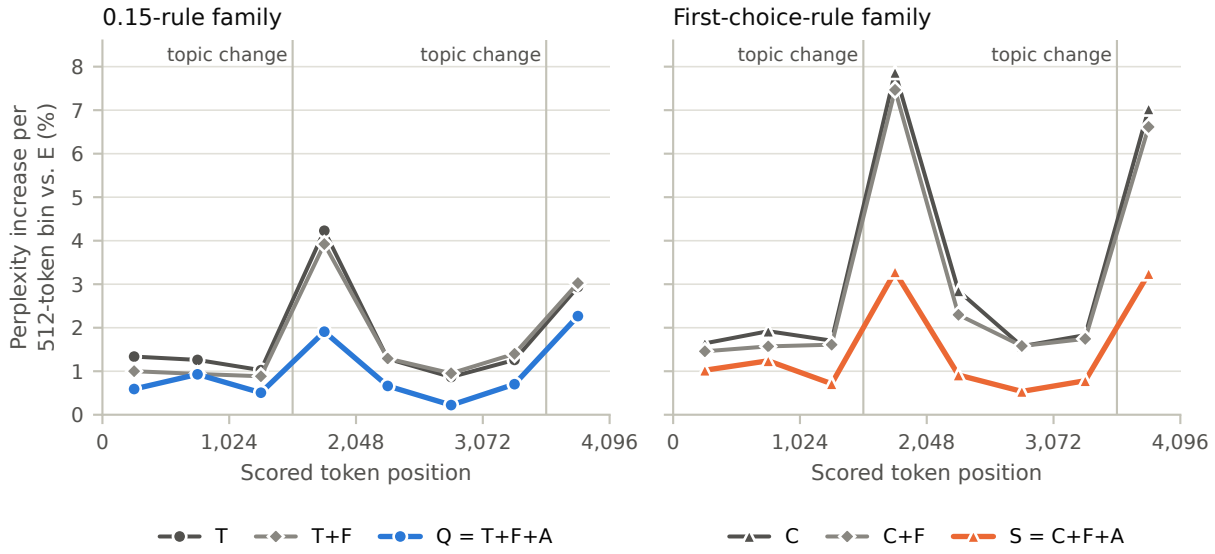


Figure 2: Loss around topic changes [exploratory · engine]. Perplexity increase over E in each 512-token bin of scored positions, pooled over the 24 topic-change sequences of R19 (25% cache; point estimates without intervals). Vertical lines mark the topic changes at scored tokens 1,536 and 3,584. Left: the 0.15-rule family (T, T+F, Q = T+F+A); right: the first-choice-rule family (C, C+F, S = C+F+A). Table F4 gives Δppl with intervals for the 512-token windows after the changes, and Table 4 the effect of admission in those windows.

weight). Among subsets with $|s(A) - m^*| \leq 0.2\,m^*$, we take the pair whose masses differ by at most $0.01\,m^*$ and whose concentrations differ the most; the heavy arm substitutes the more concentrated subset (a few heavy substitutions) and the light arm the other (many light substitutions). Where no such pair exists, neither arm substitutes. Substitution follows §3.1, and the primary condition is $m^* = 0.10$. The primary statistic is $R = \Delta\text{ppl}_{\text{heavy}}/\Delta\text{ppl}_{\text{light}}$ with a 95% interval from 2,000 session-level bootstrap resamples. If the interval for $\Delta\text{ppl}_{\text{light}}$ includes zero, we make no judgment; otherwise an interval of R within [0.8, 1.25] means that composition makes little difference, a lower bound above 1.25 that heavy substitutions are more harmful, an upper bound below 0.8 that light ones are, and an interval that merely excludes 1 gives the direction alone. The difference in Δppl between the arms and the ratio of their KL divergences were planned as auxiliary statistics. From §3.4 we expected heavy substitutions to be more harmful.

Results. In the primary condition the light arm's Δppl was +0.01% [−0.11, +0.13], so by the planned rule we made no judgment. The heavy arm's Δppl was +0.43% [+0.11, +0.77]; the planned auxiliary statistics were a difference of +0.41 [+0.13, +0.70] and a KL ratio of 1.83 [1.73, 1.93]. Both arms realized a mean mass of 0.035: a pair existed in only 33% of layer-tokens, so the realized mass was a third of the target (0.105 where substitution took place). There the heavy arm substituted 1.44 experts on average (concentration 0.80, mean rank 6.0) and the light arm 2.16 (0.48, 8.1).

Exploratory conditions. After seeing the primary result we ran three more conditions with the same code (Tables G2–G4). At a target of 0.15, a pair existed in 95% of layer-tokens and both arms realized a mass of 0.145; the heavy arm's Δppl was +4.35% [+2.76, +6.09] and the light arm's +0.49% [+0.19, +0.80], and the planned decision order would read "heavy substitutions are more harmful" (R 8.8 [6.0, 16.9]). With the slots left empty and the remaining weights renormalized (target 0.10), the arms gave +0.36% and −0.03%; at a target of 0.05, where a pair existed in only 0.9% of layer-tokens, +0.01% and −0.06%. In all four conditions, the interval of the difference between the arms lay above zero. By domain, the heavy arm lowered perplexity on math chats in all four conditions (by 0.08–2.53%); we did not investigate why.

Interpretation and limits. We reached no judgment on the planned question: the realized mass of the primary condition was small, so the light arm's loss could not be distinguished from zero and the ratio R,

with a denominator near zero, was unusable. We had not computed the fraction of layer-tokens with a pair from the model's logits before the run. Two planned auxiliary statistics (the difference and the KL ratio) and the differences in all three exploratory conditions point the same way, namely that at equal mass a few heavy substitutions cost more than many light ones; the other auxiliary statistics were less clear (R_m, like R, had an uninformative interval, and in the primary condition the heavy arm had the lower Δppl on English and math chats). The direction agrees with the exceptions in §3.4, and it suggests that mass, which followed the order of policies on the same inputs (§5.2), does not capture the effect of composition. Because the heavy arm substitutes higher-ranked experts, however, the effects of weight and rank cannot be separated. The planned decision rule also failed to handle a negative $\Delta\text{ppl}_{\text{light}}$: at a target of 0.05 it output the opposite of what the data showed ("direction only: many light substitutions are more harmful"). Further limits are that every layer-token receives the same target, that the streams of the two arms diverge after earlier substitutions, and that we used one model in BF16 emulation under teacher forcing (Appendix G).

6 Policy choice and speed

This section reports the speed and quality of the two policies we chose from the results of Section 5. Section 6.1 compares them with fetching every miss and with the engine that uses the 0.15 rule alone, on the hosts where the engine was measured, and Section 6.2 compares the engine with the 0.15 rule against stock llama.cpp on one machine. Because speed varied across hosts, every speed is reported together with the host on which it was measured. Appendix H gives the full versions of the passages shortened here.

6.1 Choosing two policies [chosen from the results]

How we chose. We screened components with the router-logit simulator (R16), measured candidates in the engine on 6 sessions and 3 topic-change sequences (R17), and chose two policies after measuring them again on 24 sessions (R18):

- quality-first: the 0.15 rule + frequency-based eviction + background admission ($\beta = 0.5$);
- speed-first: the first-choice rule + frequency-based eviction + background admission ($\beta = 0.5$).

On the 6 sessions the quality-first candidate was the 0.15 rule with frequency-based eviction; on the 24 sessions adding background admission with $\beta = 0.5$ gave the smallest perplexity increase, so we changed the candidate. The values from R18 were therefore measured on the data used for the choice. The checks after the choice are the 24 topic-change sequences (R19), which recombine the same 24 sessions in new sequences, and the generation test of §7, the only check on entirely new data. Appendix H.1 gives the simulator comparison that preceded the engine measurements.

Comparison with fetching every miss. Table 5 gives the time per token and the perplexity increase measured with teacher forcing.

Table 5: Time per token and perplexity increase relative to fetching every miss (engine, 25% cache). The ratio is the time per token of fetching every miss divided by that of each configuration; brackets give block intervals. The R18 host had an RTX 3090, 6 vCPUs (16 threads) and 5.8 GB/s O_DIRECT reads; the R19 host had an RTX 3090, 16 vCPUs (13 threads) and 5.8 GB/s O_DIRECT reads.

Configuration	Single topic, 24 sessions (R18): ms/token	Ratio	Δppl (%)	Topic change, 24 sequences (R19): ms/token	Ratio	Δppl (%)
fetching every miss	36.11	1	0	26.46	1	0
0.15 rule	12.84	2.81	+1.21 [+0.73, +1.69]	10.55	2.51	+1.77 [+1.40, +2.13]

Configuration	Single topic, 24 sessions (R18): ms/token	Ratio	Δppl (%)	Topic change, 24 sequences (R19): ms/token	Ratio	Δppl (%)
quality-first	12.89	2.80	+0.67 [+0.31, +1.04]	10.68	2.48	+0.97 [+0.67, +1.30]
speed-first	11.90	3.03	+1.00 [+0.54, +1.50]	10.08	2.62	+1.46 [+1.06, +1.87]

The two policies were 2.5–3.0× faster per token than fetching every miss, with 0.67–1.46% higher perplexity. The engine with the 0.15 rule alone was also 2.5–2.8× faster than fetching every miss, so we judge what the two policies add against the 0.15 rule. Table 6 compares them on the same run and the same tokens.

Table 6: Paired comparisons with the engine using the 0.15 rule alone (same run, same tokens): changes (%) in perplexity and in time per token, with block intervals in brackets. For topic changes, the perplexity brackets give the block interval and then the sequence interval, separated by a semicolon, followed by the number of sequences (of 24) in which the first configuration had lower perplexity.

Comparison	Single topic: perplexity	Single topic: time	Topic change: perplexity	Topic change: time
quality-first − 0.15 rule	−0.54 [−0.83, −0.25]	+0.4 [−0.5, +1.5]	−0.78 [−1.03, −0.54; −1.10, −0.47], 21/24	+1.3 [+0.4, +2.2]
speed-first − 0.15 rule	−0.20 [−0.54, +0.13]	−7.3 [−8.1, −6.5]	−0.30 [−0.57, −0.03; −0.68, +0.09], 17/24	−4.4 [−5.3, −3.4]
speed-first − quality-first	–	–	+0.49 [+0.25, +0.73; +0.19, +0.79], 5/24	−5.6 [−6.5, −4.7]

Quality-first had lower perplexity than the 0.15 rule in both data sets (−0.54% and −0.78%), and for topic changes its sequence interval was also below zero; its time per token was 0.4% and 1.3% longer (the interval for +0.4% included zero, as did the sequence-level interval [−0.5, +3.8] for +1.3%). Speed-first took 7.3% and 4.4% less time per token. Its perplexity difference had an interval including zero within a topic (−0.20%), and across topic changes only the block interval was below zero (−0.30%; sequence interval [−0.68, +0.09], sign test $p = 0.064$). For speed-first we therefore claim only that there is no evidence of higher perplexity than with the 0.15 rule, not that its perplexity is lower. Between the two policies, speed-first had 0.49% higher perplexity and took 5.6% less time per token (R19).

Results that depended on the sample and the host. In the 3 topic-change sequences each measured before the choice, speed-first had higher perplexity than the 0.15 rule (R17: +0.36% [−0.55, +1.35]; R18: +0.69% [−0.06, +1.48]); with 24 sequences the difference was −0.30%. The per-sequence values of the 24 sequences ranged from −2.32% to +1.96%, so the 3-sequence values lie largely within this variation, although the R18 value lies in its tail (about 3% of the combinations of 3 sequences had a mean of +0.69% or more). For the same 6 sessions, time per token was 24–41% longer on the R18 host than on the R17 host, and the time difference between speed-first and the 0.15 rule ranged from −4.4% to −8.3% across hosts and samples (Appendix H.1). We therefore report ratios and time differences only for the hosts measured and do not claim that they hold on other machines.

Speed in greedy generation. The generation test of §7 also recorded the time per token, which its plan designated as a reference measure to be reported separately for each pod (Table 7).

Table 7: Time per generated token in greedy generation (R20). The 0.15 rule is given in ms per token and the other configurations as changes (%) relative to it. On each pod, all six configurations generated the same problems, each problem started from an empty cache, and experts were read from the model file with O_DIRECT. Read speed is that of an O_DIRECT read of 4 GiB of the model file measured on the pod (the pods' storage was not recorded; Appendix C.5).

Pod (GPU, read speed)	Problems	0.15 rule (ms/token)	quality-first	speed-first	first-choice rule
A2 (A40, 11.3 GB/s)	200 GSM8K	15.54	−0.1	−4.6	−3.3
B (RTX 3090, 2.9 GB/s)	200 GSM8K	15.28	+0.3	−9.4	−7.1
C (A40, 11.4 GB/s)	100 GSM8K	15.24	−1.3	−8.8	−6.3
C (A40, 11.4 GB/s)	164 HumanEval	13.98	+1.3	−3.8	−3.4

In all four measurements, speed-first took 3.8–9.4% less time per token than the 0.15 rule, and quality-first was within −1.3% to +1.3%. These differences, however, include run-to-run variation: when quality-first and speed-first were rerun on the same 100 problems on the same pod, their outputs were identical token for token, yet their time per token was 3.6% and 9.5% shorter than in the original runs. The first-choice rule with a 12.5% cache (H) took 14.5–28.9% longer per token than the 0.15 rule (Appendix H.1).

Components left out of the two policies. On the 24 sessions of R18, a per-layer slot allocation gave 3.8% (0.15 rule) and 6.8% (speed-first) less mass and perplexity changes of −0.11% [−0.38, +0.17] and −0.26% [−0.53, −0.00]; our records assessed its gains as small, and we did not include it. Frequency-based eviction alone also had a small effect with the 0.15 rule (−0.26% [−0.53, +0.03] within a topic and −0.10% [−0.29, +0.08] across topic changes), although on the 6 sessions measured first its within-topic effect had an interval below zero (−0.58% [−1.09, −0.16]). Both policies use it because background admission evicts the resident expert with the lowest frequency score (Appendix H.1).

Limits.

- Tables 5 and 6 include values measured on the data used to choose the policies (R18). Of the checks after the choice, R19 reuses the same sessions in new sequences; only the test in §7 uses new data.
- In the engine we measured thresholds of 0.25, 0.5 and 1 on 6 sessions; on 24 sessions, 0.5 and 1 with the 0.15 rule and only 0.5 with the first-choice rule. We used only a half-life of 512 tokens and a frequency-score weight of 1/128. Better combinations may exist.
- One model; the two policies were measured only with a 25% cache, by teacher forcing on three RTX 3090 hosts and in the generation test on three pods (two A40s and one RTX 3090).
- The reference, fetching every miss, is the slowest configuration. The comparison with stock llama.cpp is in §6.2.

6.2 Comparison with stock llama.cpp [one machine · 0.15 rule]

Conditions. We compared the engine using the 0.15 rule with stock llama.cpp on one machine (R13): an RTX 3090 (24 GB), an NVMe SSD (Fanxiang S880, PCIe 4.0, 5.7 GB/s O_DIRECT reads) and 16 threads, with a container RAM limit of 57.7 GiB and a 45.2 GiB model file. Conditions with about 32 GB and about 16 GB of available RAM were created by other processes holding 25 and 41 GiB. The engine (25% cache, 15.7 GiB of VRAM) read missed experts either always from the SSD (SSD only) or from RAM when they were in the page cache (page cache first, the current default). Stock llama.cpp kept the experts of 14 of the 48 layers on the GPU to match our VRAM (15.8 GiB) or kept all experts on the CPU (2.8 GiB). For prompts of 128, 512 and 1,536 tokens we processed 6 sessions, one per domain, and then decoded the next 256 tokens of the data one at a time. Time to answer is the time to the first token plus the time to decode the 256 tokens, as the median over sessions; stock llama.cpp at about 32 and 16 GB was measured on the first 3 sessions only (Appendix H.2).

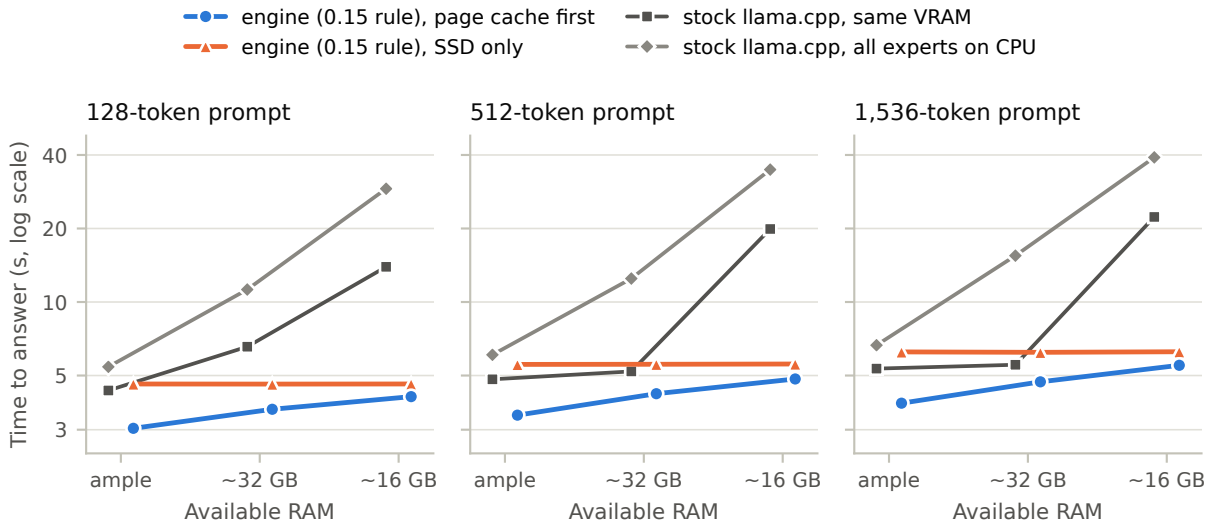


Figure 3: Time to answer on one machine (R13: RTX 3090, Fanxiang S880 SSD, 16 threads) [one machine · 0.15 rule]. Panels are prompt lengths (128, 512 and 1,536 tokens), and the x-axis is the available-RAM condition. Time to answer is the time to the first token plus the time to decode 256 data tokens one at a time (teacher forcing), reported as the median over sessions (6 sessions; stock llama.cpp at about 32 GB and 16 GB: first 3 sessions). The engine uses the 0.15 rule with a 25% cache and 15.7 GiB of VRAM. "Same VRAM" keeps the experts of 14 of the 48 layers on the GPU (15.8 GiB), and "all experts on CPU" uses 2.8 GiB. The conditions of about 32 GB and 16 GB were made by other processes holding 25 GiB and 41 GiB of the 57.7 GiB container limit, which left about 32.7 GiB and 16.7 GiB. With ample RAM, the page-cache-first value comes from the run in which the automatic switch turned that path on. To avoid overlap, stock points are drawn slightly left of each condition and engine points slightly right. Log scale; values in Table 8.

Table 8: Time to answer (s; prompts of 128 / 512 / 1,536 tokens) with decoding speed (tokens/s) in parentheses, on the same machine (R13). Ratios are the time to answer of the same-VRAM stock configuration divided by that of the engine.

RAM	Stock, all experts on CPU	Stock, same VRAM	Engine, page cache first	Engine, SSD only	Ratio: page cache first	Ratio: SSD only
ample	5.4 / 6.1 / 6.7 (59.0)	4.3 / 4.8 / 5.3 (72.9)	3.0 / 3.4 / 3.9 (108.4)[a]	4.6 / 5.6 / 6.2 (89.4)	1.4 / 1.4 / 1.4	0.94 / 0.87 / 0.85
about 32 GB	11.2 / 12.5 / 15.5 (42.9)	6.6 / 5.2 / 5.5 (65.8)	3.6 / 4.2 / 4.7 (97.8)	4.6 / 5.6 / 6.2 (89.8)	1.8 / 1.2 / 1.2	1.4 / 0.93 / 0.89
about 16 GB	29.1 / 34.9 / 39.1 (16.0)	13.9 / 19.9 / 22.3 (31.8)	4.1 / 4.8 / 5.5 (94.5)	4.6 / 5.6 / 6.3 (89.3)	3.4 / 4.1 / 4.1	3.0 / 3.6 / 3.6

[a] The run in which the automatic switch turned the page-cache-first path on.

The gap between the engine and the same-VRAM stock configuration was largest at about 16 GB (Table 8, Figure 3). In this condition the stock configuration took 3.4–4.1× (page cache first) and 3.0–3.6× (SSD only) as long as the engine to answer, and the engine decoded 89.3–94.5 tokens/s against 31.8. At about 32 GB the ratios with page cache first were 1.8 / 1.2 / 1.2 (the 128-token ratio may favor the engine: the stock configuration processed those prompts first and was slow on all three sessions, apparently while filling the page cache), and with ample RAM they were 1.4. The SSD-only path, in contrast, took 6–17% longer to answer than the stock configuration with ample RAM, and 7–12% longer for 512- and 1,536-token prompts at about 32 GB; in these two conditions the engine answered sooner at every prompt length only with page

cache first. In decoding speed alone, the engine was faster in every condition (1.2–1.5× with ample RAM and 1.4–1.5× at about 32 GB).

At about 16 GB, page cache first answered 11–13% sooner than SSD only. Measurements on another host with the same SSD model suggest that the cached pages were filled by the engine's own model loading, and when both caches were emptied before every prompt the two paths differed by about 1%; on a real 16 GB PC, where other programs use RAM, the gain may be smaller. In the same measurement the engine's perplexity was 5.5405, 1.6% higher than stock llama.cpp's (5.4526 with the same VRAM) over 4,608 tokens. When the engine fetches every miss, its perplexity differed from stock llama.cpp by only 0.01–0.04% (on an A40, R7), so the 1.6% is much larger than that difference. On another machine with slower reads (4.3 GB/s; R12), absolute values differed substantially; at about 16 GB the same-VRAM stock configuration took 4.1 / 2.6 / 3.1× as long as the engine (page cache first) to answer (Appendix H.2).

Limits.

- One machine. On another machine, absolute values and ratios differed.
- The two policies of §6.1 were not compared with stock llama.cpp; these values are for the engine with the 0.15 rule alone.
- We did not search all stock configurations (layers on the GPU, thread count, other options); the same-VRAM baseline is one configuration that matches our VRAM.
- The RAM conditions were created by other processes holding memory, which differs from memory use on a real PC.

7 A generation test planned in advance [planned]

Sections 5 and 6 measured quality by teacher-forced perplexity. This section uses a test planned in advance to check whether similar relations appear in actual generation. It asks two questions. First, does the order of mass across configurations match their order in task accuracy and in agreement with the reference configuration (fetching every miss)? Second, are the two policies of §6.1 no worse than the reference in accuracy (non-inferiority)? Hypotheses, predicted values and decision rules were fixed before the runs (Appendix E). We report every judgment, including failed predictions.

7.1 Design

Tasks and generation. We used the first 500 problems of the 1,319-problem GSM8K test set (Cobbe et al., 2021) in a seed-14 shuffle and all 164 HumanEval problems (Chen et al., 2021), each as a single user message in Qwen's chat template. Greedy decoding generates up to 1,024 tokens. Each problem starts from an empty cache: the prompt is computed exactly, its routing fills the cache, and generation follows.

Configurations and runs. The reference E fetches every miss. Q (quality-first), S (speed-first), T (the 0.15 rule alone) and C (the first-choice rule alone) use a 25% cache, and H is the first-choice rule with a 12.5% cache. Three pods split the problems: A2 (A40) took GSM8K problems 0–199, B (RTX 3090) 200–399, and C (A40) 400–499 and all of HumanEval. The six configurations of a chunk always ran on the same pod, so every paired comparison is within one pod. At the end, Q and S were run again on GSM8K problems 0–99 (Qr, Sr) to measure run-to-run differences.

Scoring, measures and statistics. A GSM8K answer is correct if the first number in the last `\boxed{}` (or, without one, the last number in the response) equals the reference answer. A HumanEval answer is correct if the code extracted from the response (the last closed Python code block, as implemented) passes the official tests within 10 s; §7.5 reports a defect in this extraction and its consequences. Agreement with E is measured as final-answer agreement (the same extracted number, or the same pass/fail outcome) and as token identity (the same generated token sequence). A problem's mass is the average over the layer-tokens generated for it. For accuracy differences (configuration − E) we use 95% percentile intervals from a problem-level paired bootstrap (2,000 resamples, seed 14) and exact McNemar tests; the non-inferiority margin is −2 points. Appendix H.3 gives the full design, including the prompts, the extraction rules and the reasons for emptying the cache.

7.2 Hypotheses and decision rules

Six hypotheses were fixed before the runs.

- **H1 (order of mass).** The mean mass during generation is ordered Q < S < T < C < H. The predicted values were about 0.07 for Q, 0.08 for S, 0.10 for T and 0.11 for C (the teacher-forced values of R18) and about 0.21 for H (simulation), with a note that short prompts might make the actual values slightly higher.
- **H2 (agreement with the reference).** For each task, across the five configurations, the Spearman correlations of mass with token identity and with final-answer agreement are both −0.9 or lower.
- **H3 (problem level).** Pooling the five configurations, the probability that a problem's final answer differs from E's increases with the problem's mass: the logistic slope is positive and its problem-level bootstrap 95% interval excludes zero.
- **H4 (accuracy).** The point estimates of the accuracy differences of Q, S, T and C lie within −1.5 to +0.5 points on both tasks, and H is at least 2 points lower on GSM8K.
- **H5 (non-inferiority).** Q and S are each non-inferior on a task if the lower bound of the 95% interval of the accuracy difference is −2 points or higher; otherwise we write "non-inferiority not shown".
- **H6 (supplementary).** The line fitted to the five configurations' (mean mass, accuracy difference) pairs has a negative slope.

Prompts, maximum length, scoring rules and the non-inferiority margin were not to be changed after seeing the results. If a defect in answer extraction were found, results under both the original and the corrected rules were to be reported.

7.3 Results

Table 9: Results by configuration (three pods pooled). Accuracy differences are configuration − E (points), with 95% intervals in brackets. Lost/gained counts problems that E solved and the configuration did not, and the reverse. Agreement is final-answer agreement and token identity with E (%).

Task	Configuration	Mass	Accuracy (%)	Difference (points)	Lost/gained	McNemar p	Final-answer agreement	Token identity
GSM8K (E 93.8%)	Q quality-first	0.158	94.2	+0.4 [−1.4, +2.0]	8/10	0.81	94.4	3.4
	S speed-first	0.168	93.6	−0.2 [−2.0, +1.4]	10/9	1.00	94.8	1.4
	T 0.15 rule	0.209	92.0	−1.8 [−3.6, 0.0]	16/7	0.093	92.8	1.4
	C first-choice rule	0.228	90.4	−3.4 [−5.8, −1.2]	25/8	0.0046	91.6	1.0
	H first-choice rule, 12.5% cache	0.370	85.4	−8.4 [−11.4, −5.6]	50/8	< 0.0001	86.6	0.4
HumanEval (E 93.9%)	Q	0.166	92.7	−1.22 [−3.05, 0.00]	2/0	0.50	98.8	36.6
	S	0.175	93.3	−0.61 [−1.84, 0.00]	1/0	1.00	99.4	38.4
	T	0.211	94.5	+0.61 [0.00, +1.83]	0/1	1.00	99.4	39.0
	C	0.225	93.3	−0.61 [−3.05, +1.22]	2/1	1.00	98.2	33.5
	H	0.369	91.5	−2.44 [−6.10, +0.61]	6/2	0.29	95.1	22.6

Table 10: Judgments under the planned decision rules.

Hypothesis	GSM8K	HumanEval
H1 order of mass	holds	holds
H1 predicted values (no decision rule in the plan)	fails (actual values 1.8–2.3× the predictions)	fails (1.8–2.4×)
H2 agreement, $\rho \leq -0.9$	holds (token identity −0.975; final answer −0.90, equal to the threshold)	fails (token identity −0.60, final outcome −0.67)
H3 problem level: judged on both tasks pooled, fails (log-odds per 0.1 of mass 0.165 [−0.048, +0.393])	per task, for reference: 0.088 [−0.137, +0.321]	per task, for reference: 0.873 [0.414, 1.499]
H4 Q, S, T, C within −1.5 to +0.5 points	fails (T −1.8, C −3.4)	fails (T +0.61)
H4 H at least 2 points lower	holds (−8.4)	not planned (−2.44)
H5 Q non-inferior	non-inferior (lower bound −1.4)	not shown (lower bound −3.05)
H5 S non-inferior	non-inferior (lower bound −2.0, equal to the margin)	non-inferior (lower bound −1.84)
H6 slope < 0	holds (−0.41 points per 0.01 of mass)	holds (−0.09 points)

On GSM8K, accuracy differences followed the order of mass. From the configuration with the least mass to the one with the most, the differences were +0.4 (Q), −0.2 (S), −1.8 (T), −3.4 (C) and −8.4 points (H) (Table 9; Figure 1c). The intervals of Q and S included zero, the interval of T had an upper bound of zero (p = 0.093), and those of C and H were below zero. Final-answer agreement also decreased in the order of mass, except that Q and S were reversed (94.4% vs. 94.8%). The fitted slope was −0.41 points per 0.01 of mass, and −0.51 without H. Almost all outputs diverged from E's (token identity 0.4–3.4%), yet 87–95% of the final answers were the same.

On HumanEval, the differences were small and the agreement prediction (H2) failed. The accuracy differences of the five configurations ranged from −2.44 to +0.61 points (Figure 1d); one problem is 0.61 points. Final-answer agreement was concentrated at 95.1–99.4%, and the Spearman correlations of mass with token identity and with final-answer agreement were −0.60 and −0.67. The fitted slope was negative (−0.09 points), but it became +0.16 points without H, so it rests on H alone.

The problem-level prediction (H3) and the predicted masses failed. The slope with both tasks pooled was 0.165 log-odds per 0.1 of mass [−0.048, +0.393]. The wording of the plan did not say whether to split by task, and we judged it on both tasks pooled; the positive slope on HumanEval alone (0.873 [0.414, 1.499]) rests on 15 cases from 10 problems, 8 of them under H. The order of mass (H1) held on both tasks, but the masses were 1.8–2.4× the predictions, which came from runs that decoded 1,535 tokens after a 512-token prompt; here each problem started from an empty cache and generated 237–328 tokens on average per configuration after a short prompt. A larger share of tokens generated before the cache filled may explain the higher values, but we did not verify this.

The two chosen policies. Quality-first was non-inferior on GSM8K (lower bound −1.4) but not shown to be non-inferior on HumanEval (lower bound −3.05; 2 problems lost, 0 gained). Speed-first was non-inferior on both tasks by the planned rule (lower bounds −2.0 and −1.84), but its GSM8K lower bound equals the margin. The point estimates of both policies were within the H4 range on both tasks; the first part of H4 failed because T and C on GSM8K and T on HumanEval were outside it. The reruns Qr and Sr were identical token for token to Q and S on all 100 problems; they ran on the same pod, so we did not measure whether this holds across pods or disk speeds.

7.4 Post-hoc analyses [exploratory]

On GSM8K, the logistic slope of answer changes on mass was negative within each of the five configurations (per 0.1 of mass, −3.1 to −7.3), and a problem's mass was negatively correlated with its generation length (−0.49 to −0.71): longer generations appear to have less mass per layer-token and to diverge from E more easily. We therefore do not use a problem's mean mass in this form as a problem-level measure. Recomputing

the planned intervals with 20 bootstrap seeds left Q's judgments unchanged, but S's HumanEval lower bound fell below the margin for 1 of 20 seeds, and with Newcombe intervals for paired differences S's lower bounds were below the margin on both tasks (-2.04 to -2.08 and -3.05 to -3.32); we therefore regard S's non-inferiority as weak evidence that holds only under the planned interval method. The GSM8K loss of C was concentrated on the problems of pod B (A2 -2.0 points, B -6.5, C 0.0), where pods and problems cannot be separated. Appendix H.3 gives the details and examples of errors.

7.5 A HumanEval scoring defect and regeneration [addendum written after the results were known]

Our HumanEval scorer accepted untagged fences as opening fences, so a non-Python block before the solution could have caused the wrong span to be extracted. We found the defect while generation was running, but the outputs had not been retained. After the original results were known, we wrote an addendum to the plan that fixed a reproduction criterion and two corrected extraction rules before any regeneration (Appendix E.4). We then regenerated all 984 HumanEval outputs (164 problems $\times$ 6 configurations) on another A40 host. All 984 records matched the original run on the six recorded fields, and re-scoring with the corrected rules changed no outcome, so all HumanEval results and judgments are the same under the original and the corrected rules. Read literally, the wording of the plan would fail one response under H that the implemented rule passes, lowering H's HumanEval accuracy from 91.5% to 90.9% and changing no judgment (Appendix H.3).

7.6 Limits

- 500 and 164 problems cannot resolve differences of about 1 point; on HumanEval one problem is 0.61 points.
- E's accuracy is near 94% on both tasks, so there is a ceiling effect.
- Each configuration ran one greedy pass, and we did not measure sampled generation. Repeated runs cover only Q and S on 100 GSM8K problems on one pod.
- Three pods with GPUs of the same generation (sm_86) took different chunks, so differences between pods and between problems are mixed. Every paired comparison is within one pod.
- The plan pooled the five configurations for H3 but did not say whether to split by task. We judged it on both tasks pooled (the analysis code also reported per-task values); judged per task, H3 would fail on GSM8K and hold on HumanEval.
- The addendum of §7.5 was written after the original results were known. Its reproduction criterion and rules were fixed before regeneration, but it does not carry the weight of a plan written before the runs.
- Emptying the cache for every problem may differ from real serving, and this test cannot say whether it is the harsher condition.
- One model and two tasks.

In summary, the order of mass across configurations held on both tasks as predicted, and on GSM8K accuracy and agreement with the reference largely followed it. Agreement on HumanEval and the problem-level relation did not match the predictions, the absolute masses were about twice the predicted values, and the predicted accuracy range for four configurations (first part of H4) failed on both tasks. Quality-first was non-inferior on GSM8K only, and speed-first's non-inferiority held only under the planned interval method.

8 Discussion and conclusion

8.1 Answers to the three questions, and the two chosen policies

Which substitutions account for most of the loss? In the emulation, most of the loss appears to come from substituting first choices or heavily weighted experts [exploratory · emulation · two models, weaker evidence from a third]. In Qwen3-Next, the 0.15 rule still substituted 7.5% of expert calls, about nine-tenths of the 8.5% when every miss was substituted, yet its perplexity increase was 0.35%, against 5.67%. In DeepSeek-V4-Flash, fetching only first choices left a loss (+1.29%) that fetching heavy misses as well reduced further (+0.20%); gpt-oss-120b showed the same order, but there the 0.15 rule also cut the substitution rate

itself, so it is weaker evidence (§5.1). The emulation is idealized, and the same 0.15 rule cost 1.21% in the engine (§5.1). These comparisons change the weight and the rank of the substituted experts together, so we do not claim that weight itself causes the loss. In Experiment B, which varied only the composition at equal mass, a few heavy substitutions also lost more than many light ones (with no judgment under its planned rule), but because weight order and rank order coincide within a token, it cannot separate weight from rank either (§5.4).

Can one quantity compare substitution policies on the same inputs? Among policies that always fetch the first choice, compared on the same inputs, the order of mass largely matched the order of perplexity increase [exploratory · engine]: the policy-level correlations were 0.92 and 0.91, with wide intervals because there were only 7 and 6 policies, and the relation had the same direction within groups of inputs (§5.2). In the planned generation test, GSM8K accuracy differences followed the order of mass, but agreement with the reference on HumanEval did not (§7). Within the scope there were exceptions in which the first-choice-rule family lost more than the 0.15-rule family at similar or smaller mass (§3.4; all exploratory, and most individual differences lie within their intervals), and in Experiment B, substitutions of equal mass lost more when their weight was concentrated on a few experts, in the auxiliary statistic and the exploratory conditions (§5.4). Between inputs, mass did not track the loss (§5.2, §7.4). We therefore use mass only as a comparative measure that roughly follows the loss order of policies on the same inputs, not as a predictor of the size of the loss or of which inputs will lose quality.

When should a cache admit an expert that keeps being substituted? A rule that accumulates the gate weight an expert loses each time it is substituted and admits it in the background at a threshold of 0.5 lowered perplexity relative to the same policies without it, most in the 512 tokens after a topic change and more for the first-choice-rule family [exploratory · engine] (§5.3). A threshold of 1 made no difference within a topic when tried with the 0.15 rule, we measured only one half-life, and our topic changes occurred in the middle of the decoded text rather than with a new prompt, which is likely harsher than a real conversation. We therefore answer this question only to the extent that one such rule lowered perplexity under the conditions we measured.

How did the two chosen policies perform? Under teacher forcing on the two hosts we measured, quality-first and speed-first were 2.5–3.0× faster per token than fetching every miss, with 0.67–1.46% higher perplexity [chosen from the results · two hosts]. Against the engine with the 0.15 rule alone, which was also 2.5–2.8× faster, quality-first had 0.54–0.78% lower perplexity at 0.4–1.3% more time per token (the interval of +0.4% included zero, as did the sequence-level interval of +1.3%), and speed-first took 4.4–7.3% less time per token with no evidence of higher perplexity (§6.1). In the generation test, the intervals of both policies' GSM8K accuracy differences included zero, quality-first was not shown to be non-inferior on HumanEval, and speed-first's non-inferiority held only under the planned interval method (§7).

8.2 Relation to prior work

Our emulation results agree in direction with the report that replacing first- or second-ranked experts degraded performance while lower-ranked replacements cost little in models with 60–64 experts per layer (Skliar et al., 2025). A benchmark with a cache of 5% of the model size found that dropping low-ranked missed experts still cost 5–30% of performance (Hoang et al., 2026); that study dropped rather than substituted missed experts and used a smaller cache, so it is not directly comparable. In our results, substituting heavy experts other than the first choice also contributed to the loss in the two models with 128 and 256 experts per layer, and in Qwen3-Next, with 512, the 0.15 rule, which protects first choices and heavy misses together, lost less than substituting every miss. In models that select 6–10 of several hundred experts, other experts than the first choice can carry large weights, so a rule that protects experts by weight as well as rank may suit them; we did not, however, compare it with rank-only thresholds other than the first-choice rule, such as protecting the top two experts.

Reports on score-based substitution differ: one study reports negligible accuracy changes for substitution thresholds below 0.35 (Zhu et al., 2026), and another found that its re-implementation of score-based substitution failed in every configuration (Hoang et al., 2026). In our emulation, the perplexity increase differed sixteenfold with similar substitution rates depending on which experts were substituted (§5.1), so whether

substitution works may depend on which experts are substituted; we have not established why the reports differ, and those studies used other models and quality measures.

We compared two other ways of updating the resident set with ours only in the simulator (§6.1, Appendix H.1): at matched estimated time per token, background admission had less mass than periodic refilling with the most frequent experts, and an approximation of SwapMoE (Kong et al., 2024) had a shorter estimated time but more mass than the 0.15 rule. These comparisons look only at mass, approximate the original method, and do not compare quality or real speed. Like TinyLFU (Einziger et al., 2017), our background admission decides what to admit separately from what to evict, and it has the form of a rent-or-buy decision (Karlin et al., 1988); because it measures the cost by gate weight and decays it, the competitive guarantees of that problem do not carry over directly, and we did not compare it with TinyLFU. Offloading studies often evaluate caches by hit rate (e.g., Wang et al., 2026); when misses are substituted, the loss differed even at similar substitution rates, but we did not compare directly whether mass or hit rate tracks the loss better.

8.3 What the results imply in practice

Within the range of our measurements, and without checking other models or machines:

- In the emulation of two models (Qwen3-Next and DeepSeek-V4-Flash), whether missed first choices and heavily weighted misses were fetched synchronously was more closely related to the loss than how many substitutions were made (§5.1), and in Experiment B, concentrated substitutions cost more than spread-out ones of equal mass in the auxiliary statistic and the exploratory conditions (§5.4).
- To choose among candidate policies, we screened them by mass measured on the same inputs and then checked a quality measure. Under teacher forcing, the simulator's predicted changes in mass between policies were within 0.1–3.6 percentage points of the engine's (§4.2). The correlation between mass and perplexity was, however, 0.77 on the 6 sessions measured first and 0.92 on 24 (§5.2); under the conditions of the generation test the masses were 1.8–2.4× the predictions (§7); and mass alone can miss the difference between the two rule families (§3.4).
- Under teacher forcing, quality-first and speed-first traded perplexity against time (on topic changes, speed-first: +0.49% perplexity and −5.6% time per token), but in the generation test both policies' accuracy intervals included zero and their order differed between the tasks (§6.1, §7).
- On one RTX 3090 machine, the engine with the 0.15 rule and the default page-cache-first path answered sooner than stock llama.cpp configured for the same VRAM in all three RAM conditions, with the largest gap at about 16 GB. There the page-cache-first path answered 11–13% sooner than the SSD-only path, but the cached pages appear to have been filled by the engine's own model loading, so the gain may be smaller on a real 16 GB PC. The SSD-only path decoded faster than stock llama.cpp in every condition but took longer to answer with ample RAM (6–17%) and for 512- and 1,536-token prompts at about 32 GB (7–12%), and the engine's perplexity was 1.6% higher (§6.2). Time per token differed by 24–41% between hosts for the same sessions (§6.1), so the ratios are values for the hosts we measured.

8.4 Limits and remaining work

- **Model and conditions.** The engine experiments use one model, Qwen3-Next-80B-A3B, mostly with a 25% cache. Quality was measured mainly by teacher-forced perplexity, and the generation test covers greedy generation on two tasks whose reference accuracy is near 94%; with 500 and 164 problems, differences of about one point are hard to resolve.
- **Grade of evidence.** Only the generation test and Experiment B were planned in advance; the rest is exploratory or chosen from the results. In the generation test, the predictions about agreement on HumanEval (H2), individual problems (H3), the accuracy range of four configurations (first part of H4) and the absolute masses did not hold (§7). Experiment B reached no judgment under its planned rule because the realized mass of its primary condition was a third of the target, and its

planned decision rule could not handle a negative loss of the light arm (§5.4). Only the generation test checked the two chosen policies on entirely new data.

- **Conclusions that changed with the sample.** Growing the sample from 6 to 24 sessions reversed two conclusions: that mass was a poor proxy (§5.2), and that background admission did not lower perplexity with the 0.15 rule (reversed for a threshold of 0.5; §5.3). The 6-session effect of frequency-based eviction weakened on 24 sessions, and with 24 new topic-change sequences the perplexity difference between speed-first and the 0.15 rule changed sign from what sets of 3 sequences had shown (§6.1). Conclusions from 24 sessions and 24 sequences may also change with larger samples.
- **Scope of comparison.** We did not compare directly with prefetching or lower-precision copies. Our only comparison with CPU computation is stock llama.cpp, made with the 0.15 rule mainly on one machine; we did not compare with CPU–GPU systems such as Fiddler (Kamahori et al., 2025) or KTransformers (Chen et al., 2025), and we did not compare the two policies with stock llama.cpp.
- **Causes.** We did not separate the effects of weight and rank (§5.1, §5.4), and we have not established why the first-choice-rule family lost more at similar or smaller mass in three measurements, most of whose differences lie within their intervals (§3.4). Experiment B is consistent with composition being one reason, but the two families also change the cache differently.

The remaining work follows from these limits: repeating the composition experiment, planned in advance, with a design whose realized mass reaches the target and a decision rule that handles negative losses, and looking for a design that separates weight from rank; problem-level analyses of the generation records that compare configurations within a problem or control for generation length, with the analysis plan written first; predicting mass in the simulator under the conditions of the generation test; comparing the two policies with stock llama.cpp on the same host and measuring topic changes that arrive with a prompt; and running the engine on a second model, combined or compared with prefetching and lower-precision copies.

8.5 Conclusion

We examined the loss from substituting missed experts when MoEs with hundreds of experts per layer are offloaded on consumer GPUs. In our emulation of two MoEs with 256 and 512 experts per layer, which experts were substituted was more closely related to the loss than how many substitutions were made. In the Qwen3-Next engine, among policies that always fetch the first choice and are compared on the same inputs, the substituted gate mass largely followed the order of perplexity and of GSM8K accuracy, but not agreement on HumanEval, and it did not indicate which inputs would lose quality. A rule that accumulates this weight and admits an expert in the background when the sum reaches 0.5 lowered perplexity, most right after topic changes, and the two policies chosen from these results had lower perplexity (quality-first) or less time per token (speed-first) than the engine with the 0.15 rule alone on the hosts we measured. Apart from the two planned experiments, these conclusions are exploratory or concern policies chosen from the results. The generation test supported our predictions only in part, and Experiment B reached no judgment under its planned rule, although in its auxiliary statistic and in exploratory conditions a few heavy substitutions lost more than many light ones of equal mass. Mass appears useful for comparing policies on the same inputs, but it does not appear to capture the effect of composition.

Use of an AI system

The author used Claude (Anthropic), a large language model, as a research assistant throughout this work. The research direction and most of the ideas came from the author, and the mathematical modeling was developed in discussions between the author and Claude. Claude wrote and maintained the code (the expert-cache patch to llama.cpp, the PyTorch emulation, the router-logit simulator and the analysis scripts) and ran the experiments on rented cloud GPUs with the author's approval. Claude wrote the first drafts of the test plans and of this paper, and the author and Claude then revised the paper together.

Numbers in this paper were taken from run logs and result files. Separate instances of Claude, given only the raw records, checked the reported numbers and statements against them, and the errors they found were corrected. Errors in our own plans and code are reported where they occur, for example the defects in the

decision rule of Experiment B (§5.4) and in the HumanEval extraction (§7.5). Claude is not an author. The author takes responsibility for the content of the paper.

Competing interests

The author declares no competing interests.

Data and code availability

The models and datasets are public (Section 4.1, Appendix C). The engine patch to llama.cpp (against commit 05af0d2b1398394cfa67e1918fee7feabccaa9bc), the emulation, the simulator, the analysis scripts and the result records will be released at `https://github.com/SOCIALPINE/moe-miss-substitution`.

References

L. A. Belady. A study of replacement algorithms for a virtual-storage computer. *IBM Systems Journal*, 5 (2):78–101, 1966. doi: 10.1147/sj.52.0078.

Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, Yuening Zhu, Qingliang Ou, Jiaqi Liao, Xianglin Chen, Zhiyuan Ai, Yongwei Wu, and Mingxing Zhang. KTransformers: Unleashing the Full Potential of CPU/GPU Hybrid Inference for MoE Models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, pp. 1014–1029. Association for Computing Machinery, 2025. doi: 10.1145/3731569.3764843.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021. URL `https://arxiv.org/abs/2107.03374`.

Yuseon Choi, Sangjin Kim, Jungjun Oh, Gwangtae Park, Byeongcheol Kim, and Hoi-Jun Yoo. Slice-MoE: Bit-Sliced Expert Caching under Miss-Rate Constraints for Efficient MoE Inference. arXiv preprint arXiv:2512.12990, 2026.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021. URL `https://arxiv.org/abs/2110.14168`.

CodeParrot. CodeParrot dataset cleaned (valid). `https://huggingface.co/datasets/codeparrot/codeparrot-clean-valid`, 2021. Accessed 2026-10-05.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. arXiv preprint arXiv:2606.19348, 2026. URL `https://arxiv.org/abs/2606.19348`.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3029–3051, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.183. URL `https://aclanthology.org/2023.emnlp-main.183/`.

Zhixu Du, Shiyu Li, Yuhao Wu, Xiangyu Jiang, Jingwei Sun, Qilin Zheng, Yongkai Wu, Ang Li, Hai Helen Li, and Yiran Chen. SiDA: Sparsity-Inspired Data-Aware Serving for Efficient and Scalable Large Mixture-of-Experts Models. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (eds.), *Proceedings of Machine Learning and Systems*, volume 6, pp. 224–238, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/698cfaf72a208aef2e78bcac55b74328-Abstract-Conference.html.

Gil Einziger, Roy Friedman, and Ben Manes. TinyLFU: A Highly Efficient Cache Admission Policy. *ACM Transactions on Storage*, 13(4):35:1–35:31, November 2017. doi: 10.1145/3149371.

Artyom Eliseev and Denis Mazur. Fast Inference of Mixture-of-Experts Language Models with Offloading. arXiv preprint arXiv:2312.17238, 2023.

ggml-org. llama.cpp: LLM inference in C/C++. https://github.com/ggml-org/llama.cpp, 2023. Accessed 2026-10-05.

Duc Hoang, Ajay Jaiswal, Mohammad Samragh, and Minsik Cho. SpecMD: A Comprehensive Study On Speculative Expert Prefetching. arXiv preprint arXiv:2602.03921, 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 *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pp. 1018–1031. IEEE, 2024. doi: 10.1109/ISCA59077.2024.00078.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=N5fVv6PZGz.

Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1–4):79–119, 1988. doi: 10.1007/BF01762111.

Byeongju Kim, Jungwan Lee, Donghyeon Han, Hoi-Jun Yoo, and Sangyeob Kim. FlashMoE: Reducing SSD I/O Bottlenecks via ML-Based Cache Replacement for Mixture-of-Experts Inference on Edge Devices. arXiv preprint arXiv:2601.17063, 2026.

Rui Kong, Yuanchun Li, Qingtian Feng, Weijun Wang, Xiaozhou Ye, Ye Ouyang, Linghe Kong, and Yunxin Liu. SwapMoE: Serving Off-the-shelf MoE-based Large Language Models with Tunable Memory Budget. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6710–6720, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.363. URL https://aclanthology.org/2024.acl-long.363/.

Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H. Noh, Sang Lyul Min, Yookun Cho, and Chong-Sang Kim. LRFU: A Spectrum of Policies that Subsumes the Least Recently Used and Least Frequently Used Policies. *IEEE Transactions on Computers*, 50(12):1352–1361, December 2001. doi: 10.1109/TC.2001.970573.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. NuminaMath. https://huggingface.co/datasets/AI-MO/NuminaMath-CoT, 2024. Dataset AI-MO/NuminaMath-CoT. Technical report: https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf. Accessed 2026-10-05.

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 *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=2XMAUP74ig.

Shuning Lin, Yifan He, and Yitong Chen. In-depth Analysis on Caching and Pre-fetching in Mixture of Experts Offloading. arXiv preprint arXiv:2511.05814, 2025.

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 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6159–6172, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.334. URL https://aclanthology.org/2024.acl-long.334/.

Vivan Madan, Prajwal Singhania, Abhinav Bhatele, Tom Goldstein, and Ashwinee Panda. Speculating Experts Accelerates Inference for Mixture-of-Experts. arXiv preprint arXiv:2603.19289, 2026.

maywell. koVast: Massive Korean multi-turn dataset. https://huggingface.co/datasets/maywell/koVast, 2024. Accessed 2026-10-05.

Nimrod Megiddo and Dharmendra S. Modha. ARC: A Self-Tuning, Low Overhead Replacement Cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*, pp. 115–130, San Francisco, CA, March 2003. USENIX Association. URL https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache.

OpenAI. gpt-oss-120b & gpt-oss-20b Model Card. arXiv preprint arXiv:2508.10925, 2025.

Qwen Team. Qwen3-Next-80B-A3B-Instruct. https://huggingface.co/Qwen/Qwen3-Next-80B-A3B-Instruct, 2025. Accessed 2026-10-05.

Arian Raje, Anupam Nayak, and Gauri Joshi. MELINOE: Fine-Tuning Enables Memory-Efficient Inference for Mixture-of-Experts Models. arXiv preprint arXiv:2602.11192, 2026.

Yuan Si, Yufeng Lin, Daming Li, and Jialu Zhang. Who Should Own the Expert Cache? Kernel-Managed Tiering for Trillion-Parameter MoE Inference. arXiv preprint arXiv:2608.12103, 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.

Xiaoniu Song, Zihang Zhong, Rong Chen, and Haibo Chen. ProMoE: Fast MoE-based LLM Serving using Proactive Caching. arXiv preprint arXiv:2410.22134, 2025.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. HOBBIT: A Mixed Precision Expert Offloading System for Fast MoE Inference. arXiv preprint arXiv:2411.01433, 2024.

Unsloth AI. Qwen3-Next-80B-A3B-Instruct-GGUF. https://huggingface.co/unsloth/Qwen3-Next-80B-A3B-Instruct-GGUF, 2025. Accessed 2026-10-05.

Yun Wang, Lingyun Yang, Senhao Yu, Yixiao Wang, Ruixing Li, Zhixiang Wei, James Yen, and Zhengwei Qi. BuddyMoE: Exploiting Expert Redundancy to Accelerate Memory-Constrained Mixture-of-Experts Inference. arXiv preprint arXiv:2511.10054, 2025.

Zihan Wang, Yuqi Wang, Lei Gong, Cheng Tang, Wenqi Lou, Teng Wang, Chao Wang, and Xuehai Zhou. SeqMoE: Toward Full-Load Performance via Predictive and Graph-Compatible MoE Offloading. arXiv preprint arXiv:2609.12978, 2026.

WiCi AI Team and SGLang Team. Running DeepSeek-V4-Flash and Kimi-K3 on consumer hardware with SSD expert pack. https://lmsys.org/blog/2026-08-29-sglang-ssd-expert-pack/, August 2026. LMSYS Blog, 29 August 2026. Accessed 2026-10-05.

Wikimedia Foundation. Wikimedia downloads. https://dumps.wikimedia.org, 2023. Wikipedia dumps of 2023-11-01 (configs 20231101.*), as the Hugging Face dataset wikimedia/wikipedia, https://huggingface.co/datasets/wikimedia/wikipedia. Accessed 2026-10-05.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen (eds.), *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL `https://aclanthology.org/2020.emnlp-demos.6/`.

Zhenhe Wu, Yaping Jin, Qinghua Xing, Hang Zhou, Wei He, Xianjie Wu, Xianfu Cheng, Jian Yang, and Hanting Chen. Cache-Aware Joint Router Adaptation for Memory-Efficient MoE Inference. arXiv preprint arXiv:2609.04895, 2026.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache. arXiv preprint arXiv:2401.14361, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. arXiv preprint arXiv:2505.09388, 2025. URL `https://arxiv.org/abs/2505.09388`.

Shuo Yang, Xiaoze Fan, Melissa Pan, Haocheng Xi, Zhe Wang, Shanlin Sun, Kurt Keutzer, Song Han, Matei Zaharia, Chenfeng Xu, and Ion Stoica. FreeToken: Efficient Edge-Native MoE Serving with Bandwidth-Adaptive Execution. arXiv preprint arXiv:2608.16157, 2026.

Yushu Zhao, Yubin Qin, Yang Wang, Xiaolong Yang, Huiming Han, Shaojun Wei, Yang Hu, and Shouyi Yin. MoBiLE: Efficient Mixture-of-Experts Inference on Consumer GPU with Mixture of Big Little Experts. In *31st Asia and South Pacific Design Automation Conference (ASP-DAC 2026)*, Hong Kong, January 2026.

Shuzhang Zhong, Ling Liang, Yuan Wang, Runsheng Wang, Ru Huang, and Meng Li. AdapMoE: Adaptive Sensitivity-based Expert Gating and Management for Efficient MoE Inference. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '24, pp. 51:1–51:9. Association for Computing Machinery, 2024. doi: 10.1145/3676536.3676741.

Guoying Zhu, Meng Li, Haipeng Dai, Xuechen Liu, Weijun Wang, Keran Li, Jun xiao, Ligeng Chen, and Wei Wang. SMoE: An Algorithm-System Co-Design for Pushing MoE to the Edge via Expert Substitution. arXiv preprint arXiv:2508.18983, 2026.

Notation used in the appendices

Runs are labeled R n, the n-th run of our research notes (Table C2). E fetches every miss synchronously and is the reference. T is the 0.15 rule, which fetches a miss synchronously if it is the router's first choice or its normalized gate weight is at least the cutoff $\tau = 0.15$; C is the first-choice rule, which fetches only the first choice. +F adds frequency-based eviction, and +A adds background admission with threshold $\beta = 0.5$ (A1: $\beta = 1$; A0.25: $\beta = 0.25$). Q = T+F+A (quality-first) and S = C+F+A (speed-first). H is C with a 12.5% cache (R20 only). Unless noted otherwise, the cache holds 25% of the experts (128 of 512 per layer). Mass is the substituted gate mass (§3.1, A.2), written as a fraction; Δppl is the perplexity increase over E on the same tokens. These letters name policies only; in §3.1 the same letters E, C, F and S denote the number of experts, the cache size and the fetched and substituted sets, which the appendices do not use.

A Engine implementation

The engine is a patch to llama.cpp (ggml-org commit 05af0d2b1398394cfa67e1918fee7feabccaa9bc). It adds a cache module and a benchmark tool and changes the context and graph code: 11 files, 2,759 added lines. The cache is enabled when a context is created and is driven only by the benchmark tool; it is not integrated into llama-server or llama-cli. Every engine build was a Release build for the GPU's compute capability. The build logs that record the CUDA toolkit (R12–R14, R18, R19 and R20s) show CUDA 12.8; the other runs did not record it. The model file was the Q4_K_M GGUF of Qwen3-Next-80B-A3B-Instruct published by unsloth (48,506,100,512 bytes, 45.2 GiB).

We kept only the final version of the patch, used in R20 (md5 48c17a90a44cca01047e5a2a5db32185). R18 and R19 ran an earlier version (md5 e4ff478b…). According to our notes, the final version only adds the greedy-generation mode of R20 and leaves the cache unchanged; because the earlier source was not kept, we cannot check this with a diff. Earlier runs used still earlier versions, whose sources were not kept either. In particular, the comparison with stock llama.cpp in A.6 used the R7 version, which read synchronous fetches with 8 threads, loaded every substituted miss in the background and had no prompt staging.

A.1 Scope and memory layout

- **What is cached.** The routed experts of all 48 MoE layers, during single-token decoding. Prompt processing is exact (A.3). The shared expert is not touched: it stays on the GPU with the dense weights, is computed every token by the stock code, is never substituted and does not enter the substituted gate mass (A.2).
- **Where the weights live.** The routed-expert tensors stay in host memory (memory-mapped file), and everything else is on the GPU (`-ngl 99 --n-cpu-moe 48`).
- **Slots.** For each MoE layer and each of its three expert weight tensors (gate, up, down), the engine allocates one GPU tensor with the file's quantization type and a slot dimension. A 512-entry map per layer gives each expert's slot. In the decode graph, the expert matrix multiplication reads the slot tensors, and the selected expert ids are mapped to slots through the map.
- **Memory.** At 25%, each layer has 128 slots for its 512 experts, and the slot tensors take 10.90 GiB. The prompt staging area (A.3) takes 2 × 0.97 GiB. Peak GPU memory (nvidia-smi) was 15.7 GiB with prompts of up to 1,536 tokens (R12–R14), and 15.2–15.5 GiB with 512-token prompts (R17–R19) and in R20. About 2.8 GiB of the 15.7 GiB is the rest of the model and the runtime state, the same as the footprint of stock llama.cpp with all experts on the CPU.
- **12.5% cache.** H in §7 and the 12.5% rows of R17 lower the number of active slots to 64 per layer, but the slot tensors stay allocated for 128 slots. No measurement in this paper therefore shows the memory saving of a 12.5% cache; the slot tensors for 64 slots would take about 5.45 GiB.

A.2 One decoding step

1. **Routing.** A callback after each MoE layer's router copies the top-K expert ids and all 512 router logits to the host. This adds one CPU–GPU synchronization per MoE layer per token (48 per token).
2. **Hits.** An expert is a hit if it is resident and fully loaded. An expert whose background read is still in flight counts as a miss and cannot serve as a substitute. If all K experts hit, the selection on the GPU is left unchanged.
3. **Fetch rule.** E fetches every miss. The 0.15 rule (T) fetches a miss if it is the router's first choice or if its gate weight, normalized over the router's top-K, is at least $\tau = 0.15$. The first-choice rule (C) fetches only the first choice.
4. **Synchronous fetch.** Victim slots are chosen first (A.4). The layer-token's fetches are then read in parallel by a fixed pool of 13 threads (12 workers and the calling thread), one task per tensor part of an expert, and the layer waits until every copy has finished.
5. **Substitution.** If at least one miss is not fetched, the final selection is the top-K by router logit among the resident experts and the experts fetched at this layer-token. The weights are recomputed by a softmax over the logits of the new selection (times the model's expert-weight scale) and written over the graph's ids and weights.

6. **Mass.** For each miss that is not fetched, the engine adds its gate weight normalized over the router's top-K. The reported mass is this sum divided by the number of decode layer-tokens, so layer-tokens without substitution count as zero. Prompt tokens are excluded.

A.3 Read paths and prompt processing

- **SSD only.** Each tensor part is read with `pread` from a file descriptor opened with O_DIRECT (a 4 KiB-aligned span) into pinned host memory and then copied to its slot. These reads bypass the page cache.
- **Page cache first.** Before each read, the engine checks with `mincore` whether every page of the part is in the page cache. If so, it copies from the memory-mapped file; otherwise it reads with O_DIRECT. This path became the default in R14. R17–R19 nevertheless read every expert with SSD only, for prompts and decoding; R20 used page cache first for prompts and for E's decoding, and SSD only for the other configurations' decoding. When the expert cache and the page cache were emptied before every prompt, the two paths differed by about 1% (R14, Table F16b).
- **Automatic switch.** At the start of prompt processing and at generation setup, one thread checks the page-cache residency of the expert tensors of every fourth MoE layer (12 layers, about 11 GB) and turns page cache first on if the resident fraction is at least 0.5. In R13 every decision within a RAM condition was the same: on at 100% residency (ample RAM), on at 67–69% (about 32 GB), off at 25–32% (about 16 GB). The check at generation setup took a median of 8 ms, 115 ms and 237 ms in these conditions. The check at the start of prompt processing ran outside the timer and was not measured.
- **Loading.** The stock loader maps the whole model file and reads all of it at load time, whereas the engine's O_DIRECT reads do not add pages to the page cache. In R14, the model file was evicted from the page cache before each run at about 16 GB, and 34.7% of it was in the page cache after the run; we attribute this to the loader.
- **Prompt processing.** The prompt is computed exactly, without substitution. For each MoE layer, every expert the prompt uses is placed in a GPU staging area: copied from its slot if resident, otherwise read from the file or the page cache. The layer's initial resident set then moves from the staging area into slots. In stock llama.cpp, the experts used by a batch of at least 32 tokens are copied to the GPU from the memory-mapped file, and pages missing from the page cache are read through page faults on a single thread.

A.4 Eviction and admission

- **Victim.** Whenever a slot is reassigned, the victim is the resident, fully loaded expert with the lowest score, excluding the router's top-K, the final selection and the experts fetched at this layer-token. Ties go to the lowest slot index.
- **Without frequency-based eviction** (T and C on their own). The score is the position of last use (LRU), updated for the router's top-K and for the substitutes actually used. The initial resident set after a prompt is the experts used most recently in the prompt, as many as there are slots (128 at 25%).
- **Frequency-based eviction (F).** For each layer, score_e(t) = Σ_s $(1-\alpha)$^(t−s) over the uses s of expert e, with $\alpha = 1/128$ in every run. A use is membership in the router's top-K at a layer-token, including experts that were then substituted; substitutes do not count. Prompt tokens count at their positions. The initial resident set is the experts with the highest score after the prompt, as many as there are slots.
- **Background admission (A).** For each substituted miss without a pending read, the engine updates an account r_e ← r_e · 2^(−Δt/512) + g_e, where Δt is the number of tokens since the account's last update (half-life 512 tokens) and g_e is the expert's normalized gate weight. If r_e ≥ β, fewer than eight admissions are pending in the layer and a victim exists, the engine sets r_e = 0, gives the victim's slot to the expert at once (the victim leaves the resident set immediately) and queues the read. Four loader threads shared by all layers serve the queue in order and pause while any synchronous read is active. The admitted expert becomes usable at its layer's first routing step

after the read has finished. The victim is the resident expert with the lowest frequency score; every configuration with admission also used F. Q and S use $\beta = 0.5$; we also measured $\beta = 1$ (A1) and $\beta = 0.25$ (A0.25).

- **Limits.** The limits on admission (fewer than eight pending per layer, and an available victim) almost never bound. No slot was ever taken from a pending admission in R17–R19. Dropped admissions and cancelled reads were zero in R17 and R18; in R19, S dropped 1 admission and cancelled 22 reads, and Q cancelled 1 read, in 98,304 decoded tokens each.
- **Volume.** With a 25% cache, completed admissions per token were 0.85–1.78 with $\beta = 1$, 2.83–3.77 with $\beta = 0.5$ on single topics, 3.99–4.83 with $\beta = 0.5$ on topic changes and 7.87 with $\beta = 0.25$ (R17); with a 12.5% cache and $\beta = 1$ they were 3.29 (R17). In R18–R19, T, C and their variants with F read 4.8–8.8 MB per token without admission and 10.4–16.4 MB with $\beta = 0.5$; E read 78–82 MB.
- **Reset.** Every configuration rebuilds the scores, slot assignments and accounts from the session's prompt record. No cache state carries over between configurations, and configurations with the same eviction scheme start from the same initial resident set.

A.5 Measurement

- **Teacher forcing (R12–R14, R17–R19).** For each session, the prompt (512 tokens in R17–R19; 128, 512 or 1,536 tokens in R12–R14) is processed once and the sequence state is saved. For each configuration, the engine restores the state, applies the configuration, fills the initial resident set and feeds the data tokens one at a time, recording the time, the NLL of the next data token and the argmax at each step. All configurations of a session run before the next session. R12–R14 measured one configuration per run.
- **Time per token** is the wall-clock time from just before a token is decoded to just after its logits are available. It includes the routing callbacks and synchronous fetches, and excludes prompt processing and the initial fill. Before the first session, each configuration decodes eight warm-up tokens from the first session, which are not measured. These normally come from the unmeasured end of that session. In the single-topic runs of R17 and R18 (512-token prompt, 1,535 scored tokens) no unmeasured end remains, so the warm-up reuses the first session's prompt and first tokens. That prompt is processed again before it is measured, so we do not expect an effect on the measured quality; the effect on timing was not measured.
- **Time to answer (R12–R14)** is the time to the first token (processing the prompt and filling the initial resident set) plus the time to decode 256 data tokens. We report the median over sessions.
- **Threads.** The benchmark used the container's CPU quota, or the number of visible CPUs when it found no quota, capped at 16: 16 threads on most hosts, 13 in R19 and 7 in R7. The R18 pod had 6 vCPUs, but the script found no quota and ran 16 threads; R20s was set to 16 threads with a quota of 7 (Table C3).
- **Greedy generation (R20).** For each problem, the engine clears the KV memory and empties the expert cache, processes the prompt exactly (page cache first), takes the first token from the prompt logits, configures the policy (initial resident set from this prompt) and decodes greedily until an end-of-generation token or 1,024 generated tokens; the count includes the first and the end token. In a local check, emptying the cache made the outputs of configurations without admission independent of problem order and chunking. Without it, the slot layout left by earlier problems changed how ties between eviction candidates were broken. Outputs with admission can still depend on when background reads finish.

A.6 Validation

- **Decision-trace replay.** On a small Qwen3-Next model (32 experts, top-4), the engine wrote every decision (fetches, substitutions, admissions and victims, with all logits) to a trace, and an independent Python implementation replayed it. All 25,317 layer-tokens with misses and all 623 background admissions matched, with both uniform and per-layer slot counts (7–28 slots), and AddressSanitizer reported no errors. The replay assumes that an admission completes before its layer's next token, so it checks the decision logic, not the timing of asynchronous reads.

- **E against stock llama.cpp.** With a small model on CPU, the outputs were bit-identical. With the real model (R7: A40, 6 sessions × 256 tokens), the engine's perplexity was 4.22565, against 4.22627 and 4.22749 for stock llama.cpp with the experts of 12 or 0 layers on the GPU (0.015% and 0.044%; the two stock configurations differ by 0.029%). The engine's value was the same at 25% and 12.5% and with both read sources of R7.
- **Read paths and prompt staging.** Perplexity was identical for SSD only, page cache first and the automatic switch at every RAM size (R13–R14; at ample RAM, R13 ran page cache first only through the automatic switch). With a small model, outputs were identical with prompt staging on and off.
- **Generation.** The HumanEval part of R20, regenerated on another host, matched the original on six recorded fields in all 984 records (E.4). The original generated texts were not kept, so they could not be compared directly.

A.7 Stock llama.cpp baselines

- The stock runs used the same benchmark binary with the cache disabled (no slots, no callback, unchanged graph), the same token files and the same teacher forcing, so per-token NLLs line up with the engine's.
- **Same VRAM.** `--n-cpu-moe 34` keeps the expert tensors of 34 of the 48 layers in host memory, so the experts of 14 layers are on the GPU. We chose this setting to match the engine's GPU memory: 15.8 GiB at peak, against 15.7 GiB for the engine. **All experts on the CPU.** `--n-cpu-moe 48`: 2.8 GiB. In both, llama.cpp computes the host-resident experts on the CPU during decoding.
- R13 used 16 threads, 6 sessions with ample RAM, and the first 3 sessions (with 2 warm-up tokens) at about 32 GB and 16 GB. These two conditions were created by other processes holding 25 GiB and 41 GiB of the 57.7 GiB container limit, which left about 32.7 GiB and 16.7 GiB. We did not tune the stock thread count, the number of expert layers on the GPU or other options.

A.8 What the engine does not do

The engine does not prefetch or predict experts during decoding (an earlier read-ahead for prompts had no effect and was not used afterwards). It does not compute experts on the CPU, does not substitute during prompt processing and keeps no pinned-RAM tier of experts. The routing decision is made on the CPU, at the cost of one synchronization per layer per token. Model loading is unchanged. Per-layer slot counts can be set only once, at startup, and slot memory is not reduced for smaller caches. The engine decodes one sequence at a time, and we did not configure it for a 16 GB GPU.

B Emulation and router-logit simulator

B.1 Emulation (R4–R5)

- **Purpose.** The emulation computes the model's outputs exactly while the cache is simulated by bookkeeping. Each policy is a separate stream of the model computed on the same tokens, next to a reference stream with the original routing.
- **Implementation.** PyTorch, layer by layer over whole sequences. The weights of a layer are loaded once (the next layer loads in the background) and used for all sessions and all streams. For Qwen3-Next and gpt-oss-120b, attention, the other token mixers and the norms are transformers modules (version 4.57.6) with weights assigned per layer, and the MoE block (router, routed experts and, for Qwen3-Next, the shared expert) is reimplemented. For DeepSeek-V4-Flash, the whole layer (hyper-connections, attention with compressed KV, norms and MoE) is reimplemented following the official inference code. Each stream carries its own residual state, so a substitution changes the inputs, and therefore the routing, of later layers and later tokens.
- **Models and numerics.**
  - Qwen3-Next-80B-A3B-Instruct in its original BF16 weights.

  - DeepSeek-V4-Flash from its FP8 weights and FP4 experts, dequantized to bf16, with the FP8 activation quantization of the official inference code emulated block by block. Its first three layers route by token hash.
  - gpt-oss-120b with its MXFP4 experts dequantized to bf16.
  - Router logits are in fp32 (from a bf16 matrix product for Qwen3-Next and gpt-oss-120b), expert outputs are accumulated in fp32, and the final log-softmax is in fp32.
- **Cache bookkeeping.**
  - The prompt (first 512 tokens) uses the original routing in every stream.
  - The resident set holds 128 or 64 experts per layer for Qwen3-Next (25% or 12.5% of 512), 64 or 32 for DeepSeek-V4-Flash (of 256) and 32 or 16 for gpt-oss-120b (of 128). It starts with the experts used most recently in the prompt.
  - At each decode token, the policy fetches or substitutes each miss as in the engine (A.2), and the weights are renormalized over the new selection.
  - Every missed expert the router chose then becomes resident from the next token, whether it was fetched or substituted (LRU replacement; the router's choices at the current token are protected). Fetch time is not modeled.
  - This residency rule is more favorable than the engine without admission, in which substituted misses are never loaded.
- **DeepSeek-V4-Flash.** The router score is s = sqrt(softplus(logit)); selection uses s plus a bias; the weights are the selected s divided by their sum, times 1.5, and a substitution renormalizes the same way over the new selection. In the three hash-routed layers every miss is fetched under every policy reported here, including "substitute every miss", so these layers have no substitutions. Substitution rates and masses are averaged over the other 40 layers.
- **Reference and metrics.** The reference is the original-routing stream of the same run, with the same code and dtypes. Each session contributes the 1,535 next-token predictions after its prompt. Δppl = exp(mean NLL of the policy − mean NLL of the reference) − 1. KL(reference ‖ policy), in nats, and argmax agreement are averaged over the same tokens.
- **Data.** 48 sessions per model: in each domain, the first eight held-out sessions (C.1), 73,680 scored tokens in total. For DeepSeek-V4-Flash and gpt-oss-120b, the sessions were rebuilt by the same procedure with each model's tokenizer and chat format, so they are not token-identical to those of Qwen3-Next. gpt-oss-120b has a very high perplexity on our data (511.8 over all domains, 2.26 on code), so we judged its quality on the code domain.
- **Validation.** All checks used tiny random models on CPU, in fp32 for Qwen3-Next and gpt-oss-120b and in bf16 for DeepSeek-V4-Flash.
  - Qwen3-Next: the layer-wise forward pass of our routing-trace tool (R2), whose token mixers, norms and weight loading the emulation reuses, matched transformers (identical routing and NLL). A test that compares the emulation itself with transformers exists, but its outcome was not recorded.
  - gpt-oss-120b: the NLL matched transformers for both plain and MXFP4 checkpoints.
  - DeepSeek-V4-Flash: we compared module by module with the official inference code, whose GPU kernels we replaced by torch equivalents written for the test. Routing, compressed KV and expert outputs were bit-identical; the remaining differences, about one bf16 ulp, come from summation order.
  - The full-size runs were in bf16 on GPUs, and the numerical differences measured by these tests were not saved.
- **Not recorded.** The GPU type and run times of R4–R5.

B.2 Router-logit simulator (R15–R16)

- **Logits.** Router logits were recorded with stock llama.cpp at the engine's base commit, from the same Q4_K_M model file, in one batched forward pass per sequence with all experts on the CPU, and stored in fp16. The logged routing is therefore the unmodified routing of this model, and a substitution cannot change later layers or later tokens in the simulator.

- **Data.**
  - all24: the 24 engine sessions, with a 512-token prompt and 1,536 decode positions each.
  - same8k: 3 sequences of 8,192 tokens, each made of four sessions of one domain.
  - cross8k: 3 sequences of 8,192 tokens from four domains, with topic changes at decode positions 1,536, 3,584 and 5,632 (7,680 decode positions). R17 ran these sequences in the engine for 4,096 positions only.
- **Replay.** Each (sequence, layer) pair is replayed independently on the CPU with the engine's rules: top-K selection, the fetch rule, substitution by logit among resident and fetched experts, mass, victim choice, LRU or frequency scores, and the admission account. Ties between victims go to the lower expert id rather than the lowest slot index. The initial resident set is filled from the prompt as in the engine.
- **Background reads.** Each layer has one background channel that completes 0.5 reads per token. An admitted expert becomes usable one token after its read ends, and admission is skipped while the channel is more than 16 tokens behind. Unlike the engine, the simulator evicts the victim when the read arrives, not when the read is scheduled.
- **Time model.**
  - Estimated time per token = 8.195 ms + 48 × (synchronous reads per layer-token) × 0.557 ms.
  - Both constants come from one R14 engine run (T, 25%, SSD only, about 16 GB of RAM, page cache emptied): 11.218 ms per token, of which 3.023 ms were spent waiting on 5.424 synchronous reads.
  - For policies with background reads, a contention factor slows synchronous reads by (1 + u), where u is the share of the token time that the SSD (5.5 GB/s; expert size 1.908 MB) spends on background reads, capped at 0.9 and solved as a fixed point. This factor proved pessimistic: for T+F+A0.25 it predicted +6.8% time against +3.5% measured (R17).
  - Background reads enter the model only through this factor, so a policy without synchronous reads always gets the floor of 8.195 ms per token.
- **Equal-time comparison.** For each policy, we varied the cutoff τ of the fetch rule (0.08, 0.11, 0.15, 0.2, 0.5) and interpolated its mass linearly at the estimated time of T ($\tau = 0.15$). If T's time lay outside a policy's range, we report the policy's fastest point.
- **Mechanisms compared in §6.1, and one further check.**
  - Periodic refill (introduced in R15; the numbers in §6.1 are from R16): every 256 decode tokens, the target set is the 128 experts with the highest frequency score ($\alpha = 1/128$, router choices only). Missing target experts stream in through the background channel, best first, each replacing the lowest-scored resident expert outside the target.
  - SwapMoE-like approximation: no synchronous fetches (every miss is substituted), and every Δ = 16, 64 or 256 tokens a refill to the 128 experts with the highest gate-weighted frequency score ($\alpha = 1/32$, 1/64 and 1/128), streamed through the same channel. This replaces SwapMoE's importance score with gate-weighted frequency, and we chose the refill periods ourselves. Without refill, the resident set stays the one built from the prompt.
  - TinyLFU-style filter: an admitted expert enters only if its frequency score exceeds the victim's. Added to T+F+A with a 25% cache, it changed nothing at the recorded precision (identical masses at $\beta = 0.5$, and at most 0.00001 apart at $\beta = 0.25$). This check is not reported in the main text.
  - In these comparisons, the refill and admission policies use frequency-based eviction, whereas T uses LRU.
- **Validation.** Against a Python reference implementation on synthetic logits, all policies agreed to five decimals, and AddressSanitizer reported no errors (the reference implementation is not in our records). Against the engine: in the R14 configuration, hit rate 84.9% vs. 84.8%, substitution 14.0% vs. 14.1% and synchronous reads 0.1076 vs. 0.1130 per layer-token; for R17, see Table F10 (the simulator on all 24 sessions, the engine on 6 of them).
- **Sensitivity to the channel.** With 0.25, 0.5 or 1 reads per token, or with arrival delays of 1, 4 or 16 tokens (one changed at a time), the mass of T+F+A on all24 ranged from 0.0726 to 0.0759.

Even the least favorable admission setting left less mass than the most favorable refill setting (0.0759 vs. 0.0816 on all24).

- **Half-life and threshold.** We ran a grid of $\beta \in \{0.1, 0.25, 0.5, 1\}$ and half-lives of 128, 512 and 2,048 tokens for T+F+A at 25%. The mass reduction saturated near $\beta = 0.25$, while background reads grew steeply as β fell; the half-life mattered mainly at $\beta = 0.5$ and 1. In the engine we then measured $\beta = 0.25$, 0.5 and 1 (R17) and kept the middle half-life, 512 tokens, in every run. Beyond these observations, we recorded no criterion for choosing 512.

C Data, runs and hosts

C.1 Sessions

Table C1: Domains. Each session is one context of 2,048 tokens.

Domain	Source (Hugging Face)	Filter	Session
ko_wiki	wikimedia/wikipedia, 20231101.ko (one of three train files)	$\geq$ 3,500 characters	one article (title and text), first 2,048 tokens
en_wiki	wikimedia/wikipedia, 20231101.en (one train file)	$\geq$ 11,000 characters	as ko_wiki
python_code	codeparrot/codeparrot-clean-valid (one file)	$\geq$ 7,000 characters, not auto-generated	one source file, first 2,048 tokens
chat_en	HuggingFaceH4/ultrachat_200k, test_sft	$\geq$ 7,000 characters	one conversation in the model's chat template, first 2,048 tokens
chat_ko	maywell/koVast (one train file)	$\geq$ 3,500 characters, $\geq$ 2 messages	as chat_en
math_chat	AI-MO/NuminaMath-CoT (first train file)	none	always packed: problem–solution pairs, each a two-message chat, concatenated to 2,048 tokens

- **Construction.** An item with at least 2,048 tokens is cut to its first 2,048 tokens, so most contexts end in the middle of a text. If a domain had too few such items, shorter items were packed back to back; how many sessions were packed was not recorded. Tokenization adds no BOS token, and chat sessions include the chat template's control tokens.
- **Seed.** Sessions were drawn with seed 0 from one random generator shared by the six domains in a fixed order.
- **Revisions.** Dataset, tokenizer and model-file revisions were not pinned, and every pod rebuilt its inputs from the Hugging Face Hub; the tokenizer and model-file commits and the model file's sha256 were logged only on the R20s pod, and R21 logged the data and model revisions (G.5). Two observations indicate that the pods built the same inputs: the six sessions shared by R17 and R18, run on different hosts, gave Δppl and argmax agreement identical to four decimals for every configuration, and the token file of the R20 tasks had the same md5 on every R20 pod, including the first attempt and R20s. We have no direct check for the other runs.
- **Sets.**
  - Pool of 178 sessions (Qwen3-Next tokenizer): 32 per domain, 18 for chat_ko.
  - Emulation (R4–R5): in each domain, the first eight sessions with an odd 0-based index (1, 3, . . . , 15; the held-out half), 48 sessions per model. For R4 these come from the pool above; for R5 the pool was rebuilt by the same procedure with each model's tokenizer and chat format (B.1).
  - Engine (R17–R19): the same procedure with 4 sessions per domain, 24 sessions. By construction these are probably the first four sessions of each domain of the pool, so 12 of them (within-domain indices 1 and 3) are probably also in the emulation set of R4. We did not check this on the token files.

- Experiment B (R21): the sessions with within-domain indices 4, 6, 8, 10, 12, 14, 16 and 17 of a rebuilt pool, 48 sessions, chosen to avoid the sets of R4 and of the engine (G.2).
- R17: the session with within-domain index 1 in each domain, 6 of the 24. R7 and R12–R14 used a file of six sessions, one per domain, whose selection script was not kept; it was made with the same generator, so these are probably the same six sessions.
- Each 2,048-token session gives 1,535 scored tokens after its 512-token prompt (the targets at 0-based positions 513–2,047).

C.2 Topic-change sequences

- **R19.** Each sequence concatenates four sessions (8,192 tokens). The engine processes a 512-token prompt and scores the next 4,096 tokens, which come from the first three sessions (1,535, 2,048 and 513 scored tokens). In 1-based counting, the domain changes at scored tokens 1,536 and 3,584.
- **Balance.** Each of the 24 sessions appears once in each of the three scored positions, the three scored parts of a sequence always come from different domains, and the 24 ordered pairs of domains at the first change are all different. Table F7 lists the domain order of every sequence.
- **Windows.** The windows "512 tokens after a change" (§5.3, Table 4) are the 0-based scored positions [1,536, 2,048) and [3,584, 4,096). They start one position after the first scored token of the new domain (0-based positions 1,535 and 3,583). The 512-token bins of Figure 2 use the same boundaries.
- **Earlier sets of three sequences** (domains of the scored parts). R17: ko_wiki → en_wiki → python_code; chat_en → chat_ko → ko_wiki; math_chat → en_wiki → chat_ko. R18: en_wiki → chat_en → math_chat; python_code → ko_wiki → chat_ko; chat_ko → math_chat → chat_en.
- **Groups.** The groups of the group-level analysis (§5.2) are the chunks in which the runs were executed. R18 has four groups of six consecutive sessions: ko_wiki ×4 + en_wiki ×2; en_wiki ×2 + python_code ×4; chat_en ×4 + chat_ko ×2; chat_ko ×2 + math_chat ×4. R19 has eight groups of three consecutive sequences.

C.3 Generation tasks (R20)

- **Problems.** GSM8K: the first 500 problems of a seed-14 permutation of the 1,319 test problems. HumanEval: all 164 problems.
- **Prompts.** One user message rendered with the model's chat template (with the generation prompt) and no system message.
  - GSM8K: the question, then on a new line "Please reason step by step, and put your final answer within $\square$."
  - HumanEval: "Complete the following Python function. Reply with the whole function, including any imports it needs, in one ```` ```python ```` code block.", a blank line, and the dataset's prompt (imports, signature and docstring) inside a ```` ```python ```` fence.
  - Prompt lengths (minimum / mean / maximum): GSM8K 46 / 83 / 180 tokens, HumanEval 74 / 170 / 427 tokens.
- **Decoding.** Greedy over the full vocabulary, at most 1,024 tokens (A.5); text is decoded without special tokens.
- **Scoring.**
  - GSM8K: the first number in the last `\boxed{}` (with matched braces); without one, the last number in the response. Commas are removed, and an answer is correct if it is within 10^{-6} of the reference answer.
  - HumanEval: the last closed code block tagged python, py or Python, or untagged; without one, everything after the last "```` ```python ````"; without that, the whole response. The prompt's import lines are prepended (and the whole prompt if the function definition is missing), the official tests and `check(entry_point)` are appended, and the program passes if it exits with code 0 within 10 seconds. A defect of this rule and its check are described in E.4.
- **Configurations and reads.** E, Q, S, T, C and H as in §7.1. For every configuration, the prompt was processed with page cache first and an empty cache. During decoding, misses were read with O_DIRECT, except in E, which read through page cache first; we do not report E's speed.

- **Chunks and pods.** GSM8K ran in chunks of 100 problems and HumanEval in two chunks of 82. Within a chunk, the configurations ran in the order E, Q, S, T, C, H. Pod A2 ran GSM8K 0–199 and, at the end, the repeats Qr and Sr on problems 0–99; pod B ran GSM8K 200–399; pod C ran GSM8K 400–499 and all of HumanEval.

C.4 Runs

Table C2: Runs used in this paper. "R n" is the n-th run of our research notes (the main text's "Part n"). R1–R3, R6 and R8–R11 were earlier explorations and are not used: R1 measured another model (Qwen3-30B-A3B), R2–R3 analysed the routing of Qwen3-Next, R6 ran a PyTorch prototype and R8–R11 ran earlier versions of the engine (B.1 cites a check of R2's tool). All runs took place between 30 September and 5 October 2026, in the order listed.

Run	What	Tool and data	Host (Table C3)	Used in
R4	Emulation of Qwen3-Next-80B-A3B	emulation; 48 sessions; 25% and 12.5%	not recorded	§3, §5.1, Table 2, Table F1
R5	Emulation of DeepSeek-V4-Flash and gpt-oss-120b	emulation; 48 sessions per model	not recorded	§3, §5.1, Table 2, Table F1
R7	E against stock llama.cpp	engine and stock; 6 sessions × 256 tokens	A40	§6.2, A.6
R12	Time to answer on another machine	engine (T) and stock; 6 sessions × prompts of 128, 512 and 1,536 tokens + 256 tokens; three RAM conditions	RTX 3090	§6.2, Table F17
R13	Time to answer, main comparison	as R12, with three read paths	RTX 3090	§6.2, Table 8, Figure 3, Table F15
R14	Page-cache residency; read paths with an emptied page cache	engine only	RTX 3090	§6.2, Table F16, B.2 (time model)
R15	Simulator, first policy search	simulator; all24, same8k, cross8k	RTX 3090 (logits), CPU	§6.1, B.2
R16	Simulator, admission and per-layer slots	simulator; same sets	RTX 3090 (logits), CPU	§5.3, §6.1, Table F9, B.2
R17	Engine, six sessions (one per domain); simulator check	engine; 6 sessions × 1,535 tokens (25% and 12.5%); 3 sequences × 4,096	RTX 3090[a]	§3, §4.2, §5.2, §5.3, §6.1, Tables F2, F8, F10–F13
R18	Engine, 24 sessions	engine; 24 sessions × 1,535 tokens; 3 sequences × 4,096; per-layer slots	RTX 3090	§3, §5.2, §5.3, §6.1, Tables 3–6, F3, F8, F12–F14, Figure 1a
R19	Engine, 24 topic-change sequences	engine; 24 sequences × 4,096 tokens	RTX 3090	§3, §5.2, §5.3, §6.1, Tables 3–6, F4–F8, F13, F14, Figures 1b and 2
R20	Generation test, planned in advance	engine, greedy generation; GSM8K 500, HumanEval 164; six configurations	2 × A40, 1 × RTX 3090	§3, §6.1 (Table 7), §7, Figures 1c–d, Tables D1, D2, F18–F20
R20s	HumanEval regeneration (addendum)	as R20, HumanEval only	A40	§7.5, E.4
R21	Experiment B: equal mass, different composition (planned in advance)	emulation (BF16, no cache); 48 sessions × 1,535 tokens, chosen to avoid earlier ones; one primary and three exploratory conditions	RTX 3090	§3 (Table 1), §5.4, Appendix G
–	Post hoc mass–loss analysis	analysis of the R18 and R19 records	–	§5.2, D.3

[a] The decision-trace replay of A.6, recorded with R17, ran on a small model on a local machine.

C.5 Hosts

Table C3: Hosts. vCPU is the pod specification and threads are the benchmark's threads. The read speed is that of `dd` reading 4 GiB of the model file in 16 MiB blocks with O_DIRECT. The RAM limit is the container's memory limit as recorded (62.0 GB = 57.7 GiB, the value used in §6.2). Every RTX 3090 had 24 GB of memory. The build logs that record the CUDA toolkit (R12–R14, R18, R19, R20s) show CUDA 12.8.

Run	GPU	Cloud tier	vCPU / threads	Storage of the model file	Read (GB/s)	RAM limit
R4–R5	not recorded	–	–	–	–	–
R7	A40 (48 GB)	secure	not recorded / 7	local NVMe, PCIe 4.0 (recorded as "PM9A3 class")	≈ 6.4[a]	50 GB
R12	RTX 3090	community	not recorded / 16	XG7000 4 TB, PCIe 4.0	4.3	62.0 GB
R13	RTX 3090	community	32 / 16	Fanxiang S880 2 TB, PCIe 4.0	5.7	62.0 GB
R14	RTX 3090	community	28 / 16	Fanxiang S880 2 TB	5.5	62.0 GB
R15–R16	RTX 3090	community	10–32 / –	– (logits recorded with experts on the CPU)	–	41–62 GB
R17	RTX 3090	community	32 / 16	XG7000 4 TB	6.1	62.0 GB
R18	RTX 3090	community	6 / 16	container disk (device not identified)	5.8	62.0 GB
R19	RTX 3090	community	16 / 13	XG7000 4 TB	5.8	62.0 GB
R20, pod A2	A40	secure	not recorded / 16	not recorded	11.3	50 GB[b]
R20, pod B	RTX 3090	community	not recorded / 16	not recorded	2.9	30 GB[b]
R20, pod C	A40	secure	not recorded / 16	not recorded	11.4	50 GB[b]
R20s	A40	secure	not recorded / 16 (CPU quota 7)	container disk	6.5	50.0 GB
R21	RTX 3090	community	32 / –	– (emulation, read from a persistent volume)	–	not recorded

[a] Not measured with `dd`. [b] From our run summary; the pod logs were not kept.

- **Storage.** The two A40 pods of R20 (A2 and C) read at 11.3–11.4 GB/s, more than any single PCIe 4.0 SSD in Table C3 (4.3–6.1 GB/s), so their storage may not be a single local NVMe device. The other A40 hosts (R7, R20s) read at about 6.4–6.5 GB/s. In our configurations, "nvme" names the read method (O_DIRECT from the model file), not a verified device.
- **Failed pods.** In R20, six pods stopped while building the engine or earlier, and none started generating; only the first left a log (E.5). In R12–R14 and R18, a few other pods were unusable and produced no results.

D Statistics

D.1 Quantities

- **NLL and Δppl.** The per-token NLL is computed in double precision over the full vocabulary. For a run, $\Delta\text{ppl} = 100 \cdot (\exp(\text{mean NLL of the configuration} - \text{mean NLL of the reference}) - 1)$, where each mean pools all scored tokens of the run; it is not an average of per-session values.
- **Time change** $= 100 \cdot (\Sigma$ ms of the configuration / Σ ms of the reference $- 1)$ over the same tokens.

- **Argmax agreement** is the share of positions whose argmax equals the reference's.
- **Pairing.** Every teacher-forced comparison between engine configurations pairs the same tokens, starting from the same saved prompt state. Most comparisons are within one run; a few pair two runs on the same tokens and host, and the tables mark them as separate runs.

D.2 Engine intervals

- **Resampling.** 2,000 resamples and 95% percentile intervals (linear interpolation).
- **Blocks.** A block is 128 consecutive scored tokens within one session or sequence. A 1,535-token session has eleven blocks of 128 tokens and one of 127; a 4,096-token sequence has 32 blocks. Each resample draws, with replacement, as many blocks as the run has from the pooled set of all its blocks (not within each session), and recomputes the statistic from summed per-token differences, so each block counts by its number of tokens. Number of blocks: R17, 72 (sessions) and 96 (sequences); R18, 288 and 96; R19, 768.
- **Sequences** (R19 only). Each resample draws the 24 sequences with replacement. For Δppl over all tokens, these intervals were 1.0–1.4 times as wide as the block intervals in comparisons with E, and 1.2–1.9 times in paired comparisons between policies.
- **Windows** after topic changes (R19) use masked sums over the same resamples. The number of window tokens varies across resamples, so the window estimate is a ratio estimator; each window is exactly four blocks.
- **Bins.** The 512-token bins of Figure 2 are point estimates without intervals.
- **Seeds.** 0, 1 and 2 for the three analysis scripts of R17, 12 for R18 and 13 for R19. R17's main script (seed 0) draws one set of block indices for each run and shares it among that run's comparisons; R17's two supplementary scripts (seeds 1 and 2) and R18 draw new indices for each comparison; in R19, one set of block indices and one set of sequence indices are shared by all comparisons.
- **Sign test** (R19). An exact two-sided binomial test with n = 24 on the number of sequences in which the first configuration had lower perplexity. No sequence had a difference of exactly zero. We computed the p-values from the logged counts (Table F5).

D.3 Mass and loss (post hoc)

- **Policy level.** Pearson r over the policies (7 in R18 and 6 in R19; E excluded), with a 95% interval from Fisher's z (standard error $1/\sqrt{(n-3)}$, z = 1.96); the least-squares slope, reported per 0.01 of mass; slopes refitted without each policy in turn; Spearman's correlation.
- **Group level.** The within-group slope after subtracting each group's mean over policies from mass and Δppl (fixed effects), with a 95% interval from 2,000 resamples of the groups (seed 14). With four groups, only 35 distinct resamples exist, so the interval is coarse. The relation between inputs is the Pearson correlation of the group means. The groups are those of C.2, and the values are in §5.2 and Table F3.

D.4 Generation test (R20)

- **Accuracy difference.** For each problem, d_i = 1[configuration correct] − 1[E correct]. The interval comes from 2,000 resamples of problems (paired) and the 95% percentile interval.
- **One random stream.** All resamples of the pooled analysis draw from one random stream (seed 14) in a fixed order: GSM8K Q, S, T, C, H; HumanEval Q, S, T, C, H; then the H3 resamples (GSM8K, HumanEval, both). An interval therefore depends on everything drawn before it from the stream. The planned judgments use the pooled analysis in this order. The per-pod analysis of pod C used the same order of comparisons but first resampled only its own 100 GSM8K problems, so its HumanEval resamples came from a different part of the stream and gave different intervals for the same data (for example, a lower bound of −3.66 instead of −3.05 for Q).
- **Interpolation.** With 2,000 resamples, the 2.5% point falls between the 50th and 51st sorted values and is interpolated. For HumanEval S these values were −2.44 and −1.83, which gives −1.84.
- **McNemar.** Exact two-sided binomial test on the problems lost (E correct, configuration wrong) and gained (the reverse).

- **H2.** Spearman's correlation over the five configurations between configuration mass and agreement. For GSM8K final answers, the sum of squared rank differences is 38, so $\rho = -0.9$ exactly, equal to the planned cutoff (-0.9); we judged with this exact value. The analysis code printed ρ without applying the cutoff, and in floating point the computed ρ is -0.8999999999999998, which a literal comparison would have judged as not meeting the cutoff.
- **H3.** Logistic regression of "final answer differs from E" on problem mass (per 0.1), over configuration $\times$ problem rows (2,500 for GSM8K, 820 for HumanEval, 3,320 for both), fitted by Newton–Raphson. The interval comes from 2,000 resamples of problems in which all five rows of a problem stay together.
- **H6.** The least-squares slope of the accuracy difference on configuration mass over the five configurations, per 0.01 of mass.
- **Agreement.** Wilson 95% intervals (Table D1). They were computed after the planned analysis, by the planned method, with a separate script.

Table D1: Agreement with E in R20: count / n (%) [Wilson 95% interval]. For HumanEval, the final answer is the pass/fail outcome.

Task	Configuration	Final answer	Token sequence
GSM8K	Q	472/500 (94.4) [92.0, 96.1]	17/500 (3.4) [2.1, 5.4]
	S	474/500 (94.8) [92.5, 96.4]	7/500 (1.4) [0.7, 2.9]
	T	464/500 (92.8) [90.2, 94.8]	7/500 (1.4) [0.7, 2.9]
	C	458/500 (91.6) [88.8, 93.7]	5/500 (1.0) [0.4, 2.3]
	H	433/500 (86.6) [83.3, 89.3]	2/500 (0.4) [0.1, 1.4]
HumanEval	Q	162/164 (98.8) [95.7, 99.7]	60/164 (36.6) [29.6, 44.2]
	S	163/164 (99.4) [96.6, 99.9]	63/164 (38.4) [31.3, 46.0]
	T	163/164 (99.4) [96.6, 99.9]	64/164 (39.0) [31.9, 46.7]
	C	161/164 (98.2) [94.8, 99.4]	55/164 (33.5) [26.8, 41.1]
	H	156/164 (95.1) [90.7, 97.5]	37/164 (22.6) [16.8, 29.5]

The repeats Qr and Sr (GSM8K problems 0–99) agreed with Q and S on all 100 problems in both measures ([96.3, 100.0]).

D.5 Sensitivity of the non-inferiority judgments (post hoc)

Table D2: Lower bound of the accuracy difference from E (points). The margin is -2.

Task, configuration	Planned (seed-14 stream)	20 seeds (0–19), one stream per comparison	Newcombe, paired, Wilson-based: ϕ uncorrected / corrected
GSM8K, Q	-1.4	-1.4 to -1.2	-1.37 / -1.41
GSM8K, S	-2.0	-2.0 to -1.8	-2.04 / -2.08
HumanEval, Q	-3.05	-3.05 for every seed	-4.01 / -4.24
HumanEval, S	-1.84	-1.83 for 19 seeds, -2.44 for seed 16	-3.05 / -3.32

- **Why HumanEval S varies.** S lost one problem and gained none, so a resample's difference is $-K/164$, where K is the number of times that problem is drawn. $P(K \geq 4) = 0.019$, so the exact 2.5% quantile of the resampling distribution is -1.83. In the planned stream, exactly 50 of the 2,000 resamples were at -2.44 or below.
- **Counts used** (both correct / lost / gained / both wrong): GSM8K Q 461/8/10/21, S 459/10/9/22; HumanEval Q 152/2/0/10, S 153/1/0/10.
- **Newcombe variants.** The two columns differ only in whether the correlation term ϕ uses Newcombe's correction. We report both because we could not confirm which one corresponds to the published method. Under both, the lower bound of S is below the margin on both tasks, and that of Q on GSM8K is above it.

E Plan of the generation test

E.1 Plan and timeline

We wrote the plan of the generation test in Korean before any generation. One section ("Execution change 1") was added to it before generation, and an addendum on the HumanEval extraction rule was written after the original results were known (E.4). The plan of Experiment B is described in Appendix G. The hypotheses, predictions and decision rules have not changed since before any generation. The analysis and task-building code that we kept is identical to what the pods received once the same docstring removal is applied, and the run script and the engine patch are identical as they are (Table E1).

Table E1: Checksums of the code sent to the pods, and where they were printed.

File	md5	Printed by
Archive sent to pods A2, B and C	4e1972a59a7e58b8f561aaefd0f61e30	all three pods
Analysis code (docstrings removed, as sent)	d347c08f207898d786984afb7a248750	all three pods and the regeneration pod
Engine patch	48c17a90a44cca01047e5a2a5db32185	every R20 pod, including the first attempt and R20s

Timeline (3 October 2026, UTC): - Before 06:56 The plan is written. - 06:56 A first attempt starts. Its engine build fails because github.com cannot be resolved, and no problem is generated. - 07:48 Execution change 1 is added to the plan (file modification time 07:48:41; the section header gives 07:50). - 07:55–08:01 Pods B, C and A2 are created, each with the plan including Execution change 1. - 08:05–08:11 Generation starts (pod C 08:05, pods B and A2 08:11). - 10:49 We notice the HumanEval extraction defect while pod C is generating HumanEval with T. - 11:07–11:56 The pods finish (A2 11:06:55, C 11:10:57, B 11:56:16). They are deleted at about 11:20 (A2, C) and 12:11 (B). The per-problem records, with an md5 on every line, were kept; the generated texts were not. - The pooled analysis follows; its time is not recorded. - 14:31 The addendum is saved, after the pooled results were known. - 14:41–16:36 HumanEval is regenerated (R20s); first generation at 14:50:52, scoring at 16:36.

E.2 What the plan fixed

- **Questions.** Does mass also explain task accuracy and agreement with E in real generation? Are the two chosen policies non-inferior to E?
- **Design.** Qwen3-Next-80B-A3B-Instruct (Q4_K_M) with a 25% cache, experts read from the model file with O_DIRECT (the plan says "NVMe"), exact prompt processing, a community-cloud RTX 3090, and an engine that adds only a greedy generation mode.
- **Tasks, prompts and generation.** As in C.3 and A.5, including an empty cache for every problem (with its reason), page cache first for E, and no speed reported for E. The plan quotes the GSM8K instruction word for word and describes the HumanEval prompt in words; the exact HumanEval string is in the task-building code.
- **Configurations.** E, Q, S, T, C and H (§7.1). Chunks form the outer loop and configurations the inner loop; Q and S are repeated at the end on the first GSM8K chunk.
- **Metrics.** Correctness: GSM8K as in C.3; HumanEval by the wording of the plan quoted in E.5 item 2, which the implementation in C.3 follows except in three respects. Agreement with E (final answer, and identity of the token sequence) and problem mass. Generation length, truncation and time per token for reference only.
- **Hypotheses H1–H6** (§7.2). The predicted masses for H1 were Q $\approx$ 0.07, S $\approx$ 0.08, T $\approx$ 0.10 and C $\approx$ 0.11 (from the teacher-forced values of R18) and H $\approx$ 0.21 (simulator), with the remark that actual values "may be a little higher" because prompts are short. No decision rule was given for these values.

- **Analysis.** Paired bootstrap over problems (2,000, seed 14) and exact McNemar tests for accuracy differences; Wilson intervals for agreement; a logistic regression with a problem-level bootstrap for H3. Every number is reported, including failed predictions.
- **Not to be changed.** Prompts, maximum length, scoring rules and the −2-point margin. If an extraction defect is found, results under both the original and the corrected rule are reported.
- **Known limits.** An expected paired standard error of 0.5–1 point, a ceiling near 95% accuracy, run-to-run variation from background reads (measured by the repeats), and one model on one device.

E.3 Execution change 1 (appended before generation)

The section states that only the execution changed, before any results were seen; hypotheses, predictions, decision rules, metrics, prompts and configurations did not. After the first attempt failed, the work was split over three pods to shorten the total run time: GSM8K 0–199 and the repeats (A; the pod was named A2), GSM8K 200–399 (B), and GSM8K 400–499 and HumanEval (C). All six configurations of a chunk run on one pod, so every paired comparison and the repeats stay within a pod. Because RTX 3090 pods were in short supply, any GPU of the Ampere generation with compute capability 8.6 is allowed (RTX 3090, RTX 3090 Ti, A40, RTX A6000), and speed is reported per pod, never pooled. Each pod analyzes its own chunks and prints per-problem records with an md5 on every line, and the final result pools these records once with the same code; the pooled analysis found no md5 mismatch among 4,184 records. The limits line became "one model, three pods with GPUs of the same generation".

E.4 Addendum and regeneration

- **Defect.** The extraction pattern in the analysis code, which was fixed before generation (md5 d347c08f…), accepts an untagged fence as an opening fence. If a non-Python block precedes the solution, the closing fence of that block can therefore be read as an opening fence. We noticed this at 10:49 UTC, while generation was still running and before the pooled analysis. The pods were nevertheless deleted after they finished without keeping the generated texts, so the original responses could not be rescored.
- **What the addendum fixed** (written after the pooled results were known and before the regeneration):
  - Regenerate HumanEval (164 problems × 6 configurations) under pod C's conditions, with the same files and order.
  - Reproduction criterion: every record must match the original on six recorded fields (number of tokens, end-of-generation flag, mass × 10^4 rounded, correctness under the original rule, final-answer agreement and token identity).
  - Rule A (decision): read fences line by line and take the last closed block tagged python, py or Python, or untagged; otherwise fall back to the original rule. Rule B (reference): as A, but accept only the tags python, py and python3, ignoring case.
  - Recompute the HumanEval judgments under rule A, side by side with the original.
- **Outcome.** All 984 records matched on all six fields, which meets the reproduction criterion of the addendum (E.5 item 11). No problem's outcome depended on the rule (original, A or B), and every planned judgment was unchanged. All 982 opening fences found in the 984 regenerated responses were tagged python, and no response showed the defect pattern.

E.5 Departures from the plan and other points

1. **Scope of H3.** The plan pooled the five configurations but did not say whether to pool the two tasks. The analysis code, fixed before generation, printed three scopes, and we judged on both tasks pooled: 0.165 [−0.048, +0.393], so H3 did not hold. By task: GSM8K 0.088 [−0.137, +0.321], HumanEval 0.873 [0.414, 1.499].
2. **HumanEval extraction: wording and implementation.** The wording of the plan takes the last python code block; if the block is unclosed, everything after the last "```` ```python ````"; if there is no block, the whole response. The implementation in the analysis code fixed before generation

differs in three ways. (a) It also accepts py, Python and untagged fences. (b) It takes the last closed block whenever one exists, ignoring a trailing unclosed block. (c) It has the fence-pairing defect (E.4). (a) and (c) changed no outcome. (b) matters for two H responses truncated at 1,024 tokens. HumanEval/93 fails either way. HumanEval/76 passes as implemented but fails under the literal wording; under that reading, H's HumanEval accuracy is 90.85% (difference −3.05 [−6.71, 0.0], lost/gained 7/2, McNemar p = 0.18), and H3 is 0.924 [0.484, 1.546] for HumanEval and 0.173 [−0.037, +0.396] pooled. No planned judgment changes. Rule B is not a literal reading of the wording either: it accepts py and python3 and keeps the preference for closed blocks.

3. **Wilson intervals** were in the plan, but the analysis code fixed before generation does not compute them. We computed them afterwards with a separate script by the planned method (Table D1).
4. **Pods and code.** Execution change 1 allowed A40 GPUs but did not mention the secure cloud, on which both A40 pods ran. Between the first attempt and the main run, the analysis code and the run script also changed, consistent with the per-pod analysis and pooling of Execution change 1. The earlier analysis code was not kept, so we cannot show the difference.
5. **Failed pods.** Six pods failed before generation, at the engine build or earlier, and only the first left a log. Which version of the plan the other five carried is not recorded.
6. **H1.** The plan gave no decision rule for the predicted masses. We judged the prediction not met because the measured masses were 1.8–2.4 times the predictions.
7. **H4 for H** was planned only for GSM8K; we report the HumanEval value (−2.44) as not planned.
8. **H2 on GSM8K** final answers equals the planned cutoff exactly (D.4).
9. **Seed.** "Seed 14" alone does not fix the intervals, which depend on everything drawn before them from the stream: the order of the comparisons and the number of problems in each (D.4). The judgments use the pooled analysis in its default order, which Execution change 1 specifies as the single pooled computation; the addendum's recomputation used the same order.
10. **Regeneration conditions.** The threads were set to 16 as written, but the container's CPU quota allowed 7. The host differed, the disk read 6.5 GB/s instead of 11.4 GB/s, and the original container image was not recorded. The regeneration script also changed logging and the default number of repeats (to 0, as on pod C).
11. **What the reproduction covers.** The six recorded fields, not the token sequences themselves, which were not kept.
12. **Status of the addendum.** It was written after the original results were known; only the regeneration came after it. We therefore do not give it the weight of a plan made before a run (§7.6).
13. **Not deviations.** The repeats ran at the end of pod A2, about two hours after Q and S, as planned. The mean GSM8K prompt (83 tokens) was shorter than the "about 100–300 tokens" mentioned in the remark on H1, which is not a rule.

F Additional results

Each table supports statements in the main text, which its caption names; notation is as at the start of the appendices. Brackets are 95% intervals: block intervals, or block; sequence intervals. "Separate run" marks comparisons between two runs on the same tokens and host. Evidence grades follow the corresponding sections of the main text (§4.4); the post hoc analyses (D.3, D.5, Table F20) and the values marked ‡ are exploratory. Values marked ‡ are statistics that we computed from the logged per-token, per-sequence or per-problem arrays while preparing this appendix; all other values are as logged or follow from logged values by arithmetic (differences, ratios, sums, weighted means and p-values of exact tests).

F.1 Emulation (§5.1)

Table F1: Emulation with a 12.5% cache (R4–R5; LRU; 48 sessions, 73,680 scored tokens per model), relative to the original model. For DeepSeek-V4-Flash, the substitution rate and mass exclude the three hash-routed layers. At 12.5%, the first-choice rule with substitution was not run for either model. Supports §5.1 (the paragraph on the 12.5% cache) and §3.3–3.4.

Model (reference)	Policy	Δppl (%)	KL (nats)	Argmax agreement (%)	Top-1 accuracy (%)	Substitution rate (%)	Mass
Qwen3-Next-80B-A3B (ppl 3.920; top-1 68.94%)	substitute every miss	+18.8	0.250	85.5	65.78	20.0	0.185
	0.15 rule	+1.62	0.033	94.2	68.68	17.2	0.132
DeepSeek-V4-Flash (ppl 3.371; top-1 72.32%)	substitute every miss	+13.9	0.213	86.6	69.80	29.0	0.250
	0.15 rule	+0.28	0.029	94.6	72.22	19.0	0.124

F.2 Mass and loss in the engine (§5.2)

Table F2: The six sessions of R17 (one per domain, within-domain index 1; 6 × 1,535 scored tokens). Changes in parentheses are relative to T at the same cache size. Times are comparable only within a run; C+F was measured in a separate run. Supports §5.2 (correlation 0.77 on these sessions), §3.4 (second exception) and §6.1.

Cache	Configuration	Tokens/s	ms/token	Δppl (%)	Argmax agreement	Hit rate (%)	Mass	MB/token	Admissions/token
25%	E	37.2	26.89 (+165.2%)	0	1.000	91.0	0	82.4	0
	T	98.6	10.14	+1.40 [+0.59, +2.35]	0.949	85.5	0.1012	7.8	0
	C	103.0	9.71 (−4.2%)	+1.75 [+0.86, +2.66]	0.940	84.5	0.1149 (+14%)	5.7	0
	T+F	99.1	10.10 (−0.5%)	+0.81 [+0.14, +1.53]	0.953	86.2	0.0955 (−6%)	7.6	0
	T+F+A1	98.0	10.20 (+0.6%)	+1.00 [+0.39, +1.64]	0.954	87.8	0.0834 (−18%)	9.5	0.86
	Q	96.8	10.33 (+1.9%)	+1.12 [+0.57, +1.71]	0.958	89.0	0.0751 (−26%)	14.0	3.11
	T+F+A0.25	95.3	10.49 (+3.5%)	+1.04 [+0.50, +1.60]	0.957	89.9	0.0682 (−33%)	23.1	7.87
	C+F (separate run)	102.3	9.78 (−3.6%)	+1.44 [+0.58, +2.42]	0.943	85.5	0.1073 (+6%)	5.4	0
	C+F+A1	104.1	9.60 (−5.3%)	+1.12 [+0.34, +1.96]	0.948	87.6	0.0915 (−10%)	7.6	1.15
	S	103.6	9.65 (−4.8%)	+0.86 [+0.23, +1.47]	0.951	88.8	0.0820 (−19%)	12.7	3.77
12.5%	T	75.8	13.19	+2.85 [+1.79, +3.91]	0.922	71.5	0.1983	19.3	0
	C	84.1	11.89 (−9.9%)	+4.87 [+3.62, +6.13]	0.907	70.1	0.2236 (+13%)	13.5	0
	C+F+A1	80.5	12.42 (−5.9%)	+3.61 [+2.49, +4.77]	0.917	75.2	0.1842 (−7%)	21.2	3.29

Over the seven policies of Table 3 (T, C, T+F, T+F+A1, Q, C+F and S), the Pearson correlation between mass and Δppl was 0.77 on these six sessions and 0.92 on all 24.

Table F3: Groups of six consecutive sessions in R18 (C.2): mass / Δppl (%) of each policy. The summary below the table covers both R18 and R19 (eight groups of three sequences). Supports §5.2 and §3.3 (condition 2).

Policy	Group 1: ko_wiki ×4, en_wiki ×2	Group 2: en_wiki ×2, python_code ×4	Group 3: chat_en ×4, chat_ko ×2	Group 4: chat_ko ×2, math_chat ×4
T	0.0771 / +2.40	0.0952 / +1.24	0.0652 / +0.34	0.1535 / +0.87
C	0.0890 / +2.97	0.1081 / +1.83	0.0718 / +0.81	0.1716 / +1.46
T+F	0.0715 / +1.93	0.0890 / +0.92	0.0602 / +0.33	0.1514 / +0.60
T+F+A1	0.0639 / +1.86	0.0779 / +0.75	0.0532 / +0.50	0.1259 / +0.69
Q	0.0574 / +1.80	0.0703 / +0.67	0.0483 / −0.04	0.1108 / +0.25
C+F	0.0812 / +2.75	0.1000 / +1.56	0.0657 / +0.57	0.1677 / +1.35
S	0.0629 / +2.57	0.0765 / +1.04	0.0522 / +0.33	0.1201 / +0.10
Group mean‡	0.0719 / +2.33	0.0881 / +1.14	0.0595 / +0.40	0.1430 / +0.76
Spearman (mass, Δppl)	0.786	0.893	0.821	0.929
Perplexity of E	4.648	3.567	4.160	3.817

Summary, R18 / R19: Pearson r 0.918 [0.533, 0.988] / 0.909 [0.371, 0.990]; slope per 0.01 of mass 0.246 / 0.260 (0.208–0.268 / 0.219–0.296 with one policy left out); Spearman 0.857 / 1.000; within-group slope 0.233 [0.200, 0.312] / 0.263 [0.196, 0.316]; Spearman within groups 0.786–0.929 / 0.829–1.000; correlation of the group means −0.209 / −0.318; correlation across groups within each policy −0.445 to −0.018 / −0.299 to −0.046.

F.3 Topic changes (§5.3)

Table F4: Loss by position in the 24 topic-change sequences (R19): Δppl (%). "After change 1" and "after change 2" are the 512-token bins that follow each change (Figure 2). "Windows" pools both bins, and "rest" is the other 3,072 scored tokens of each sequence. Supports §5.3.

Policy	After change 1	After change 2	Windows	Rest	Whole sequence (Table 3)
T	+4.2	+2.9	+3.58 [+2.51, +4.79; +2.53, +4.70]	+1.17 [+0.85, +1.49; +0.84, +1.51]	+1.77
C	+7.9	+7.0	+7.45 [+5.83, +9.13; +5.43, +9.33]	+1.92 [+1.53, +2.31; +1.48, +2.39]	+3.27
T+F	+3.9	+3.0	+3.47 [+2.34, +4.65; +2.11, +4.81]	+1.08 [+0.80, +1.34; +0.72, +1.42]	+1.67
Q	+1.9	+2.3	+2.09 [+1.22, +3.03; +1.35, +2.84]	+0.60 [+0.32, +0.88; +0.10, +0.97]	+0.97
C+F	+7.5	+6.6	+7.04 [+5.28, +8.94; +4.66, +9.43]	+1.71 [+1.33, +2.08; +1.19, +2.21]	+3.02
S	+3.3	+3.2	+3.26 [+2.07, +4.55; +2.17, +4.31]	+0.87 [+0.52, +1.23; +0.36, +1.37]	+1.46

For T, the windows hold 25% of the scored tokens and 50.2%‡ of the increase in log perplexity.

Table F5: Paired comparisons on the 24 topic-change sequences (R19): change in perplexity and in time per token (%). "Lower" is the number of sequences (of 24) in which the first configuration had lower perplexity, with the p-value of the two-sided sign test. Supports §5.3, Table 4 and Table 6.

Comparison	Whole sequence	Lower (p)	Windows	Lower	Rest	Time
T+F − T	−0.10 [−0.29, +0.08; −0.37, +0.17]	13 (0.84)	−0.10 [−0.60, +0.36; −0.69, +0.49]	11	−0.09 [−0.28, +0.09; −0.30, +0.11]	+0.5 [−0.1, +1.1; −0.4, +1.6]
Q − (T+F)	−0.69 [−0.93, −0.43; −1.18, −0.24]	20 (0.002)	−1.34 [−2.00, −0.59; −2.32, −0.36]	17	−0.47 [−0.68, −0.23; −0.89, −0.09]	+0.8 [+0.2, +1.5; −0.6, +2.4]
Q − T	−0.78 [−1.03, −0.54; −1.10, −0.47]	21 (< 0.001)	−1.44 [−2.10, −0.73; −2.20, −0.66]	18	−0.56 [−0.79, −0.33; −0.87, −0.29]	+1.3 [+0.4, +2.2; −0.5, +3.8]
C − T	+1.48 [+1.13, +1.85; +0.95, +1.98]	4 (0.002)	+3.73 [+2.63, +4.94; +2.29, +5.06]	5	+0.74 [+0.47, +0.99; +0.44, +1.05]	−4.5 [−5.1, −3.9; −5.6, −3.1]
C+F − C	−0.25 [−0.49, −0.01; −0.56, +0.03]	15 (0.31)	−0.38 [−1.14, +0.35; −1.13, +0.45]	13	−0.20 [−0.40, +0.00; −0.46, +0.00]	+0.6 [−0.1, +1.4; −0.3, +1.9]
S − (C+F)	−1.51 [−1.85, −1.18; −2.07, −0.95]	20 (0.002)	−3.53 [−4.55, −2.46; −5.01, −2.07]	20	−0.83 [−1.07, −0.58; −1.21, −0.40]	−0.6 [−1.4, +0.5; −1.5, +0.6]
S − C	−1.75 [−2.10, −1.42; −2.15, −1.35]	22 (< 0.001)	−3.89 [−4.87, −2.95; −5.06, −2.72]	22	−1.03 [−1.33, −0.73; −1.32, −0.75]	+0.1 [−0.7, +1.0; −1.6, +2.1]
S − T	−0.30 [−0.57, −0.03; −0.68, +0.09]	17 (0.064)	−0.31 [−1.06, +0.44; −1.02, +0.43]	17	−0.30 [−0.56, −0.04; −0.70, +0.05]	−4.4 [−5.3, −3.4; −6.3, −1.9]
C+F − T	+1.23 [+0.87, +1.57; +0.63, +1.82]	5 (0.007)	+3.34 [+2.16, +4.56; +1.77, +4.94]	3	+0.53 [+0.28, +0.79; +0.14, +0.91]	−3.9 [−4.6, −3.1; −5.5, −1.5]
S − Q	+0.49 [+0.25, +0.73; +0.19, +0.79]	5 (0.007)	+1.15 [+0.44, +1.87; +0.56, +1.80]	3	+0.26 [+0.05, +0.49; −0.05, +0.59]	−5.6 [−6.5, −4.7; −6.6, −4.6]

Table F6: Spread of the per-sequence values over the 24 topic-change sequences (R19), %: median; minimum to maximum; standard deviation, all computed by us from the logged per-sequence values‡. "Lower" as in Table F5. Supports §6.1 ("a conclusion that changed with the sample").

Quantity	Whole sequence	Lower	Windows	Lower
T vs E	+1.65; +0.31 to +3.95; 0.97	–	–	–
C vs E	+3.47; −0.01 to +6.41; 1.68	–	–	–
Q vs E	+1.08; −2.38 to +2.49; 1.06	–	–	–
S vs E	+1.44; −0.58 to +3.89; 1.29	–	–	–
Q − (T+F)	−0.69; −4.31 to +1.76; 1.19	20	−1.57; −6.42 to +4.78; 2.50	17
Q − T	−0.86; −2.86 to +0.99; 0.80	21	−2.15; −5.16 to +3.27; 1.93	18
S − (C+F)	−1.81; −4.89 to +1.32; 1.39	20	−3.66; −9.76 to +5.54; 3.86	20
S − C	−2.03; −3.63 to +0.53; 1.00	22	−3.77; −8.59 to +3.02; 3.05	22
S − T	−0.22; −2.32 to +1.96; 0.96	17	−0.61; −4.79 to +4.57; 1.85	17
S − Q	+0.44; −1.26 to +1.93; 0.75	5	+1.20; −2.57 to +4.20; 1.56	3
C − T	+1.80; −1.59 to +3.43; 1.30	4	+4.70; −3.30 to +7.91; 3.42	5

Over all 2,024 subsets of three sequences‡, the mean of the three per-sequence values of S − T was at least +0.36 (the R17 value) in 10.7% of the subsets and at least +0.69 (the R18 value) in 2.9%. For S − (C+F), the mean was at least −0.76 (R18) in 15.7% and at most −2.23 (R17) in 15.5%; for S − C, it was at least −0.62 (R18) in 2.5%.

Table F7: Per-sequence values in R19 (%): Δppl of four policies and three paired differences. The domains are those of the three scored parts (C.2): koW ko_wiki, enW en_wiki, code python_code, chatE chat_en, chatK chat_ko, math math_chat. Supports §6.1 and Table F6.

Sequence	Domains	T	C	Q	S	S − T	Q − T	S − T, windows
0	koW → enW → chatE	+1.13	+3.55	+0.71	+0.98	−0.15	−0.42	−1.38
1	koW → code → chatK	+2.10	+4.56	+1.14	+2.25	+0.14	−0.94	−0.88
2	koW → chatK → code	+1.53	+3.43	+0.83	+1.46	−0.07	−0.69	−0.02
3	koW → math → enW	+2.27	+3.06	+2.24	+3.19	+0.90	−0.03	+0.99
4	enW → code → chatK	+3.95	+6.41	+2.49	+3.89	−0.05	−1.41	−0.29
5	enW → chatE → math	+1.60	−0.01	+0.90	−0.37	−1.94	−0.69	−1.34
6	enW → math → chatE	+0.79	+0.06	+0.74	+0.59	−0.20	−0.05	−0.37
7	enW → koW → code	+3.00	+5.06	+1.83	+3.48	+0.47	−1.13	+0.57
8	code → chatE → math	+0.82	+0.24	−0.48	−0.08	−0.89	−1.29	−2.52
9	code → chatK → koW	+1.98	+3.74	+1.61	+1.89	−0.09	−0.36	−0.27
10	code → koW → chatK	+3.25	+3.51	+2.26	+2.50	−0.73	−0.96	−1.24
11	code → enW → chatE	+3.54	+6.25	+2.07	+2.95	−0.57	−1.43	−1.64
12	chatE → chatK → koW	+1.22	+3.23	+0.01	+0.67	−0.54	−1.19	−0.98
13	chatE → math → enW	+1.78	+1.23	+0.04	−0.58	−2.32	−1.72	−1.06
14	chatE → enW → math	+0.58	+3.92	+1.02	+2.55	+1.96	+0.44	+4.57
15	chatE → code → chatK	+2.19	+4.26	+1.45	+1.94	−0.24	−0.72	−1.50
16	chatK → math → enW	+0.31	+1.96	+1.30	+0.97	+0.66	+0.99	+2.94
17	chatK → koW → code	+1.70	+3.32	+0.87	+1.43	−0.26	−0.81	−0.02
18	chatK → code → koW	+1.84	+3.39	+0.91	+1.14	−0.69	−0.91	−0.95
19	chatK → chatE → math	+1.21	+2.38	−0.27	−0.21	−1.41	−1.46	−4.79
20	math → koW → code	+1.26	+4.73	+1.29	+2.46	+1.18	+0.04	+1.22
21	math → enW → chatE	+2.61	+3.64	+1.44	+1.16	−1.41	−1.14	−0.85
22	math → chatE → enW	+0.49	+3.25	−2.38	−0.50	−0.99	−2.86	+1.13
23	math → chatK → koW	+1.42	+3.72	+1.42	+1.50	+0.08	−0.00	+1.74

Table F8: Paired comparisons on topic-change sequences in three runs: change in perplexity (%). R17 and R18 used three sequences each (C.2) and report block intervals; R19 used 24 sequences (block; sequence intervals). Supports §6.1 and §8.4.

Comparison	R17 (3 sequences)	R18 (3 sequences)	R19 (24 sequences)
T+F − T	−0.07 [−0.64, +0.54] (separate run)	+0.24 [−0.26, +0.84]	−0.10 [−0.29, +0.08; −0.37, +0.17]
C − T	+2.46 [+1.42, +3.58]	+1.32 [+0.40, +2.29]	+1.48 [+1.13, +1.85; +0.95, +1.98]
C+F − C	+0.18 [−0.49, +0.92] (separate run)	+0.15 [−0.80, +1.08]	−0.25 [−0.49, −0.01; −0.56, +0.03]
S − (C+F)	−2.23 [−3.14, −1.27] (separate run)	−0.76 [−1.85, +0.35]	−1.51 [−1.85, −1.18; −2.07, −0.95]
S − C	−2.05 [−3.10, −1.06]	−0.62 [−1.43, +0.23]	−1.75 [−2.10, −1.42; −2.15, −1.35]
S − T	+0.36 [−0.55, +1.35]	+0.69 [−0.06, +1.48]	−0.30 [−0.57, −0.03; −0.68, +0.09]
C+F − T	–	+1.47 [+0.25, +2.78]	+1.23 [+0.87, +1.57; +0.63, +1.82]
Q − T	not run	not run	−0.78 [−1.03, −0.54; −1.10, −0.47]
T+F+A1 − T	−0.53 [−1.27, +0.25]	+0.09 [−0.41, +0.63]	not run

Weighted by the number of sequences, the 30 sequences give about −0.1% for S − T and about −1.5% for S − (C+F).

F.4 Choosing the two policies (§6.1)

Table F9: Router-logit simulator with a 25% cache (R16): changes relative to T on all24 / same8k / cross8k (B.2). Time per token is the model's estimate with contention. "At equal time" is the change in mass when τ is varied so that the estimated time equals T's. These are masses only; they imply neither quality nor measured speed (§8.2). Supports §6.1, §5.3 and §8.2.

Policy	Mass (%)	Estimated time (%)	Background reads (MB/token)	Mass at equal time (%)
C	+12 / +16 / +21	−5.4 / −4.9 / −5.5	0	–
No synchronous fetch, no refill	+114 / +220 / +387	−19.3 / −17.1 / −21.8	0	–
SwapMoE-like, refill every 16 / 64 / 256 tokens	+34 / +31 / +82; +14 / +3 / +1; +32 / +27 / +41	−19.3 / −17.1 / −21.8	9.8–43.6	–
T + periodic refill every 256 tokens (introduced in R15)	−16 / −23 / −27	+4.7 / +5.5 / +6.4	7.5–8.1	−9 / −15 / −18
T+F+A1	−18 / −24 / −28	+0.7 / +0.8 / +1.9	1.6–2.5	–
Q = T+F+A	−25 / −32 / −38	+3.3 / +2.9 / +4.5	5.6–7.3	−22 / −30 / −34
T+F+A0.25	−29 / −37 / −42	+6.8 / +6.0 / +8.2	13.0–15.5	−23 / −32 / −36
C+F+A1	−10 / −17 / −21	−5.4 / −4.7 / −5.0	2.1–3.3	–
S = C+F+A	−18 / −26 / −31	−3.7 / −3.3 / −3.4	6.6–8.6	–

T itself had masses of 0.0972, 0.1019 and 0.1335, estimated times of 10.15, 9.89 and 10.47 ms per token, and hit rates of 86.1%, 85.1% and 80.6%. The policies without synchronous fetches get the model's floor time by construction (B.2), and refilling every 16 tokens schedules close to the channel's capacity of 45.8 MB per token.

Table F10: Simulator against engine: changes relative to T at the same cache size. The simulator side uses all 24 sessions (R16) and the engine side 6 of them (R17). Supports §4.2 and §8.3.

Cache	Configuration	Mass change: simulator / engine (difference, points)	Time change: simulator / engine (difference, points)	Hit rate (%): simulator / engine
25%	T (absolute)	0.0972 / 0.1012	10.15 / 10.14 ms	86.1 / 85.5
	C	+12.3% / +13.6% (1.3)	−5.4% / −4.2% (1.2)	85.3 / 84.5
	T+F	−5.2% / −5.6% (0.4)	−0.5% / −0.5% (0.0)	86.8 / 86.2
	T+F+A1	−17.6% / −17.5% (0.1)	+0.7% / +0.6% (0.1)	88.4 / 87.8
	Q	−24.8% / −25.8% (1.0)	+3.3% / +1.9% (1.4)	89.3 / 89.0
	T+F+A0.25	−29.0% / −32.6% (3.6)	+6.8% / +3.5% (3.3)	89.9 / 89.9
	C+F+A1	−10.2% / −9.6% (0.6)	−5.4% / −5.3% (0.1)	88.1 / 87.6
	S	−18.1% / −18.9% (0.8)	−3.7% / −4.8% (1.1)	89.1 / 88.8
12.5%	T (absolute)	0.1886 / 0.1983	13.08 / 13.19 ms	73.0 / 71.5
	C	+12.2% / +12.7% (0.5)	−11.8% / −9.9% (1.9)	71.8 / 70.1
	C+F+A1	−7.3% / −7.1% (0.2)	−6.7% / −5.9% (0.8)	76.6 / 75.2

Table F11: Paired comparisons in R17: change in perplexity and in time per token (%), block intervals. Perplexity is reproducible across runs, time is not (Table F13b). Rows marked * are cited in §3.4, §5.3 or §6.1.

Data	Comparison	Perplexity	Time	Same run?
6 sessions, 25%	T+F − T *	−0.58 [−1.09, −0.16]	−0.5 [−1.0, +0.1]	yes
	T+F+A1 − (T+F)	+0.18 [−0.12, +0.50]	+1.1 [+0.6, +1.7]	yes
	Q − (T+F) *	+0.30 [−0.15, +0.76]	+2.4 [+1.4, +3.4]	yes
	T+F+A0.25 − (T+F)	+0.23 [−0.25, +0.68]	+3.9 [+3.0, +5.0]	yes
	C+F − C	−0.30 [−0.77, +0.17]	+0.6 [−0.3, +1.6]	separate
	S − (C+F)	−0.57 [−1.21, +0.10]	+1.7 [+0.1, +3.4]	yes (second run)
	S − C	−0.87 [−1.65, −0.12]	−0.6 [−1.5, +0.2]	yes
	C+F+A1 − C	−0.61 [−1.19, −0.06]	−1.1 [−1.9, −0.4]	yes
	C − T	+0.34 [−0.42, +0.94]	−4.2 [−5.0, −3.5]	yes
	S − T *	−0.53 [−1.30, +0.14]	−4.8 [−5.6, −4.0]	yes
	C+F+A1 − T	−0.27 [−1.11, +0.40]	−5.3 [−6.1, −4.5]	yes
	Q − T *	−0.28 [−0.88, +0.26]	+1.9 [+0.7, +3.2]	yes
3 topic-change sequences, 25%	T+F − T	−0.07 [−0.64, +0.54]	−0.7 [−1.2, −0.2]	separate
	T+F+A1 − (T+F)	−0.46 [−1.13, +0.21]	+1.0 [+0.6, +1.5]	separate
	C+F − C	+0.18 [−0.49, +0.92]	−0.1 [−0.6, +0.3]	separate
	S − (C+F)	−2.23 [−3.14, −1.27]	+0.9 [−0.1, +1.8]	separate
	C − T	+2.46 [+1.42, +3.58]	−5.4 [−6.1, −4.7]	yes
	S − T *	+0.36 [−0.55, +1.35]	−4.8 [−5.6, −3.9]	yes
	C+F+A1 − T	+0.81 [−0.12, +1.81]	−6.1 [−6.6, −5.5]	yes
	T+F+A1 − T	−0.53 [−1.27, +0.25]	+0.3 [−0.3, +0.9]	yes
6 sessions, 12.5%	C − T	+1.96 [+1.13, +2.78]	−9.9 [−10.7, −8.9]	yes
	C+F+A1 − C	−1.20 [−2.16, −0.24]	+4.4 [+3.3, +5.6]	yes
	C+F+A1 − T *	+0.73 [−0.13, +1.64]	−5.9 [−6.8, −4.9]	yes

Q was not measured on these three sequences.

Table F12a: Paired comparisons in R18 beyond Table 6: change in perplexity (%) on all 24 sessions, on the 18 sessions not used in R17, and on R17's six sessions, and change in time per token (%) on each host. Block intervals. The column for R17's six sessions gives R17's own values and intervals; on these sessions R18 gave the same point estimates. Supports §6.1 ("Components left out of the two policies"), §5.3 ($\beta = 1$) and §8.4.

Comparison	24 sessions	18 new sessions	R17's 6 sessions	Time, 24 sessions (R18 host)	Time, 6 sessions (R17 host)
T+F − T	−0.26 [−0.53, +0.03]	−0.16 [−0.48, +0.18]	−0.58 [−1.09, −0.16]	+0.1 [−0.7, +1.0]	−0.5 [−1.0, +0.1]
T+F+A1 − (T+F)	+0.00 [−0.19, +0.19]	−0.06 [−0.30, +0.16]	+0.18 [−0.12, +0.50]	−0.5 [−1.2, +0.3]	+1.1 [+0.6, +1.7]
Q − (T+F)	−0.28 [−0.53, −0.04]	−0.47 [−0.78, −0.18]	+0.30 [−0.15, +0.76]	+0.4 [−0.6, +1.2]	+2.4 [+1.4, +3.4]
C+F − C	−0.21 [−0.46, +0.04]	−0.17 [−0.47, +0.12]	−0.30 [−0.77, +0.17] (separate run)	−1.1 [−1.8, −0.3]	separate run
S − (C+F)	−0.54 [−0.84, −0.26]	−0.53 [−0.88, −0.20]	−0.57 [−1.21, +0.10]	−0.4 [−1.1, +0.4]	+1.7 [+0.1, +3.4]
S − C	−0.75 [−1.12, −0.38]	−0.71 [−1.15, −0.30]	−0.87 [−1.65, −0.12]	−1.4 [−2.2, −0.6]	−0.6 [−1.5, +0.2]

Comparison	24 sessions	18 new sessions	R17's 6 sessions	Time, 24 sessions (R18 host)	Time, 6 sessions (R17 host)
C − T	+0.55 [+0.22, +0.88]	+0.62 [+0.24, +0.98]	+0.34 [−0.42, +0.94]	−6.0 [−6.6, −5.3]	−4.2 [−5.0, −3.5]
C+F − T	+0.34 [+0.01, +0.68]	+0.44 [+0.05, +0.83]	+0.04 [−0.70, +0.67] (separate run)	−7.0 [−7.8, −6.2]	separate run
T+F+A1 − T	−0.26 [−0.55, +0.04]	−0.21 [−0.57, +0.16]	−0.40 [−0.99, +0.09]	−0.4 [−1.3, +0.6]	+0.6 [−0.2, +1.5]
Q − T (Table 6)	−0.54 [−0.83, −0.25]	−0.62 [−0.96, −0.30]	−0.28 [−0.88, +0.26]	+0.4 [−0.5, +1.5]	+1.9 [+0.7, +3.2]
S − T (Table 6)	−0.20 [−0.54, +0.13]	−0.09 [−0.48, +0.29]	−0.53 [−1.30, +0.14]	−7.3 [−8.1, −6.5]	−4.8 [−5.6, −4.0]

Table F12b: Per-layer slot counts at equal VRAM (R18, 24 sessions). The counts (86–259 slots per layer; 10.89 GiB in total, against 10.90 GiB for 128 slots everywhere) are the average of the allocations computed from the prompts of the same 24 sessions, so they were built from the measured sessions. Time changes compare separate runs. Supports §6.1.

Policy	Slots	Tokens/s	ms/token	Hit rate (%)	Mass	MB/token	Δppl (%)	Change vs. uniform: perplexity (%)	Change vs. uniform: time (%)
T	uniform	77.9	12.84	86.0	0.0978	7.0	+1.21 [+0.73, +1.69]	–	–
T	per layer	79.7	12.54	86.5	0.0940 (−3.8%)	6.0	+1.09 [+0.73, +1.46]	−0.11 [−0.38, +0.17]	−2.3 [−3.1, −1.5]
S	uniform	84.0	11.90	89.3	0.0779	11.8	+1.00 [+0.54, +1.50]	–	–
S	per layer	85.7	11.67	90.0	0.0726 (−6.8%)	10.4	+0.74 [+0.29, +1.20]	−0.26 [−0.53, −0.00]	−1.9 [−2.7, −1.1]

In R17 (six sessions), per-layer slots reduced the mass by 4.2% (T) and 7.2% (S); the simulator had predicted 4.6% and 6.9%.

Table F13a: The same six sessions on the R17 host (32 vCPUs) and the R18 host (6 vCPUs). Δppl and argmax agreement were identical to four decimals on both hosts. Supports §5.2 (measurement noise) and §6.1 (host-dependent speed).

Configuration	Δppl (%), both hosts	Argmax agreement, both hosts	ms/token: R17 → R18 host	Change
E	0	1.0000	26.89 → 37.92	+41%
T	+1.4010	0.9492	10.14 → 13.30	+31%
C	+1.7480	0.9395	9.71 → 12.37	+27%
T+F	+0.8142	0.9526	10.10 → 13.24	+31%
T+F+A1	+0.9998	0.9539	10.20 → 13.27	+30%
Q	+1.1199	0.9579	10.33 → 13.45	+30%
C+F (separate run in R17)	+1.4420	0.9429	9.78 → 12.11	+24%
S	+0.8623	0.9509	9.65 → 12.20	+26%

Table F13b: Rerunning a configuration on the same host (R17). Supports §5.2 and §6.1.

Configuration	First run (ms/token)	Second run (ms/token)	Time change (%)	Perplexity change (%)
S, 6 sessions	9.65	9.94	+3.0 [+1.8, +4.5]	+0.00 [−0.06, +0.07]
Q, 6 sessions	10.33	10.56	+2.1 [+0.8, +3.9]	+0.01 [−0.01, +0.04]
S, 3 topic-change sequences	10.09	9.89	−1.9 [−2.6, −1.3]	+0.04 [−0.06, +0.14]

Table F13c: Time differences between configurations by host and sample (%), block intervals. Supports §6.1.

Measurement	Host: vCPU (threads), read speed	S − T	Q − T	Q − (T+F)
R17, 6 sessions	32 (16), 6.1 GB/s	−4.8 [−5.6, −4.0]	+1.9 [+0.7, +3.2]	+2.4 [+1.4, +3.4]
R18, 24 sessions	6 (16), 5.8 GB/s	−7.3 [−8.1, −6.5]	+0.4 [−0.5, +1.5]	+0.4 [−0.6, +1.2]
R18, R17's 6 sessions	6 (16), 5.8 GB/s	−8.3 [−9.7, −6.9]	+1.1 [−0.8, +3.0]	+1.6 [+0.1, +3.1]
R18, 3 topic-change sequences	6 (16), 5.8 GB/s	−4.9 [−6.5, −3.4]	not run	not run
R19, 24 topic-change sequences	16 (13), 5.8 GB/s	−4.4 [−5.3, −3.4]	+1.3 [+0.4, +2.2]	+0.8 [+0.2, +1.5]

S − Q was −5.6 [−6.5, −4.7] in R19 (paired); from the values relative to T, it was about −7.7 in R18 and −6.6 in R17 (values relative to E, as in §6.1, give the same ratio).

Table F14: Teacher-forced decoding with a 25% cache in R18 (24 sessions) and R19 (24 sequences). Changes in parentheses are relative to T; intervals for Δppl are in Table 3. Supports §5.3 and §6.1.

Data	Configuration	Tokens/s	ms/token	Δppl (%)	Argmax agreement	Hit rate (%)	Mass	MB/token	Admissions/token
R18, 24 sessions	E	27.7	36.11 (+181.3%)	0	1.000	91.5	0	78.2	0
	T	77.9	12.84	+1.21	0.956	86.0	0.0978	7.0	0
	C	82.8	12.07 (−6.0%)	+1.76	0.948	85.2	0.1101 (+13%)	5.1	0
	T+F	77.8	12.85 (+0.1%)	+0.94	0.958	86.7	0.0930 (−5%)	6.8	0
	T+F+A1	78.2	12.79 (−0.4%)	+0.95	0.960	88.4	0.0802 (−18%)	8.7	0.85
	Q	77.6	12.89 (+0.4%)	+0.67	0.961	89.5	0.0717 (−27%)	13.0	2.97
	C+F	83.7	11.94 (−7.0%)	+1.55	0.950	86.0	0.1037 (+6%)	4.8	0
	S	84.0	11.90 (−7.3%)	+1.00	0.955	89.3	0.0779 (−20%)	11.8	3.56
R19, 24 sequences	E	37.8	26.46 (+150.8%)	0	1.000	91.1	0	81.6	0
	T	94.8	10.55	+1.77	0.936	80.6	0.1337	8.6	0
	C	99.3	10.07 (−4.5%)	+3.27	0.922	78.1	0.1600 (+20%)	6.5	0
	T+F	94.3	10.60 (+0.5%)	+1.67	0.938	80.7	0.1327 (−1%)	8.8	0

Data	Configuration	Tokens/s	ms/token	Δppl (%)	Argmax agreement	Hit rate (%)	Mass	MB/token	Admissions/token
	Q	93.6	10.68 (+1.3%)	+0.97	0.950	87.4	0.0851 (−36%)	16.4	3.99
	C+F	98.6	10.14 (−3.9%)	+3.02	0.922	78.2	0.1587 (+19%)	6.5	0
	S	99.2	10.08 (−4.4%)	+1.46	0.942	87.2	0.0929 (−31%)	15.1	4.78

Relative to the same policy without admission, admission reduced the mass by 22.9% (Q vs T+F) and 24.8% (S vs C+F) in R18, and by 35.9% and 41.5% in R19.

F.5 Comparison with stock llama.cpp (§6.2)

Table F15a: One machine (R13; Table 8, Figure 3): time to answer (s) for prompts of 128, 512 and 1,536 tokens, with time to first token and decoding time in parentheses; session medians. Stock llama.cpp at about 32 GB and 16 GB: first 3 sessions. "Worst case": the expert cache and the page cache were emptied before every prompt. Supports §6.2.

RAM	Configuration	128	512	1,536	Decoding tokens/s	VRAM (GiB)	Load (s)
ample	stock, all experts on CPU	5.4 (1.1 / 4.3)	6.1 (1.8 / 4.3)	6.7 (2.3 / 4.3)	59.0	2.8[a]	3.1
	stock, same VRAM	4.3 (0.8 / 3.5)	4.8 (1.3 / 3.5)	5.3 (1.8 / 3.5)	72.9	15.8[a]	4.2
	engine, automatic (on; 100% resident)	3.0 (0.6 / 2.4)	3.4 (1.1 / 2.3)	3.9 (1.5 / 2.3)	108.4	15.7	3.1
	engine, SSD only	4.6 (1.7 / 3.0)	5.6 (2.7 / 2.8)	6.2 (3.5 / 2.7)	89.4	15.7	3.1
about 32 GB	stock, all experts on CPU	11.2 (4.9 / 6.4)	12.5 (7.2 / 5.3)	15.5 (10.5 / 5.7)	42.9	2.8	7.3
	stock, same VRAM	6.6 (2.1 / 4.4)	5.2 (1.5 / 3.7)	5.5 (2.0 / 3.6)	65.8	15.8	21.5
	engine, automatic (on; 67–69% resident)	3.6 (1.0 / 2.6)	4.1 (1.6 / 2.5)	4.7 (2.2 / 2.5)	99.1	15.7	11.0
	engine, SSD only	4.6 (1.7 / 3.0)	5.6 (2.7 / 2.8)	6.2 (3.5 / 2.7)	89.8	15.7	19.2
	engine, page cache first	3.6 (1.0 / 2.7)	4.2 (1.6 / 2.6)	4.7 (2.2 / 2.5)	97.8	15.7	20.0
about 16 GB	stock, all experts on CPU	29.1 (11.3 / 16.9)	34.9 (17.5 / 17.4)	39.1 (20.9 / 18.2)	16.0	2.8	19.6
	stock, same VRAM	13.9 (5.9 / 8.0)	19.9 (11.6 / 8.6)	22.3 (13.1 / 8.8)	31.8	15.8	24.1
	engine, automatic (off; 25–32% resident)	4.8 (1.9 / 2.9)	5.8 (3.0 / 2.8)	6.5 (3.8 / 2.8)	89.5	15.7	20.3
	engine, SSD only	4.6 (1.7 / 3.0)	5.6 (2.7 / 2.8)	6.3 (3.5 / 2.7)	89.3	15.7	17.2
	engine, page cache first	4.1 (1.3 / 2.8)	4.8 (2.2 / 2.7)	5.5 (2.9 / 2.6)	94.5	15.7	20.1
	engine, automatic, worst case	5.6 (2.7 / 2.9)	6.9 (4.1 / 2.8)	7.8 (5.1 / 2.8)	89.8	15.7	17.5

[a] Not logged with ample RAM; the values are those of the same settings at about 32 GB and 16 GB.

Table F15b: Ratios of the time to answer, stock ÷ engine, for prompts of 128 / 512 / 1,536 tokens (two decimals‡). Supports §6.2 and Table 8.

RAM	Engine read path	Same-VRAM stock ÷ engine	All-CPU stock ÷ engine
ample	automatic (on)	1.43 / 1.40 / 1.39	1.8 / 1.8 / 1.7
	SSD only	0.94 / 0.87 / 0.85	–

RAM	Engine read path	Same-VRAM stock ÷ engine	All-CPU stock ÷ engine
about 32 GB	page cache first	1.80 / 1.23 / 1.17	–
	automatic (on)	1.82 / 1.26 / 1.18	3.1 / 3.0 / 3.3
	SSD only	1.42 / 0.93 / 0.89	–
about 16 GB	page cache first	3.40 / 4.11 / 4.05	7.1 / 7.2 / 7.1
	automatic (off)	2.89 / 3.43 / 3.43	6.0 / 6.0 / 6.0
	SSD only	3.02 / 3.57 / 3.57	–
	automatic, worst case	2.50 / 2.90 / 2.84	5.2 / 5.1 / 5.0

If the unmeasured check at the start of each prompt costs as much as the check at generation setup (0.24 s), the automatic switch at about 16 GB gives about 2.8 / 3.3 / 3.3. Computed on the same three sessions as stock, the ratios change by +0.7% to +3.2% (worst case up to +7.2%), and at about 32 GB with 512-token prompts SSD only is 3.8% slower than stock. Engine perplexity on the same tokens was 5.5405, against 5.4526 (stock, same VRAM) and 5.4538 (stock, all experts on CPU).

Table F16a: Page-cache residency of the 45.2 GiB model file on another host with the same SSD model (R14; engine only). Supports §6.2.

Point	Resident share
Ample RAM, before and after the runs	100% / 100%
Right after RAM was limited to about 16 GB	36.4% (then emptied to 0.0%)
After a page-cache-first run	34.7%, about 15.7 GiB (then emptied to 0.0%)
After an SSD-only run	34.7% (then emptied to 0.0%)
After a second page-cache-first run	34.7% (then emptied to 0.0%)
After the two worst-case runs	3.4%

Loading took 16.5–19.0 s at about 16 GB and 3.2 s with ample RAM.

Table F16b: Time to answer on the R14 host (s; time to first token / decoding time), session medians. Supports §6.2.

Condition	Read path	128	512	1,536	Decoding tokens/s
ample	page cache first	3.0 (0.6 / 2.3)	3.4 (1.1 / 2.3)	3.8 (1.5 / 2.3)	111.2
	SSD only	4.7 (1.8 / 2.9)	5.7 (2.9 / 2.8)	6.3 (3.6 / 2.7)	90.5
about 16 GB, page cache emptied before the run	page cache first	4.0 (1.3 / 2.7)	4.7 (2.1 / 2.6)	5.4 (2.8 / 2.6)	96.2
	SSD only	5.1 (2.0 / 3.1)	5.7 (2.9 / 2.8)	6.4 (3.7 / 2.7)	89.1
	page cache first, again	4.0 (1.3 / 2.7)	4.8 (2.1 / 2.6)	5.4 (2.8 / 2.6)	95.0
about 16 GB, worst case	page cache first	5.5 (2.6 / 2.9)	6.8 (4.0 / 2.8)	7.7 (5.0 / 2.7)	90.1
	SSD only	5.5 (2.6 / 2.9)	6.7 (4.0 / 2.8)	7.7 (5.0 / 2.7)	91.0

Table F17: Another machine with slower disk reads (R12; XG7000, 4.3 GB/s): time to answer (s; time to first token / decoding time), session medians. The engine used the 0.15 rule and page cache first. Stock llama.cpp at about 32 GB and 16 GB: first 3 sessions. Supports §6.2 (“another machine”).

RAM	Configuration	128	512	1,536	Decoding tokens/s
ample	stock, all experts on CPU	5.8 (1.2 / 4.6)	6.5 (1.9 / 4.6)	7.1 (2.5 / 4.6)	55.9
	stock, same VRAM	4.6 (0.9 / 3.7)	5.1 (1.4 / 3.7)	5.7 (2.0 / 3.7)	69.4

RAM	Configuration	128	512	1,536	Decoding tokens/s
about 32 GB	engine	3.2 (0.7 / 2.5)	3.6 (1.2 / 2.4)	4.0 (1.6 / 2.4)	104.2
	stock, all experts on CPU	17.0 (7.2 / 10.0)	17.7 (11.2 / 6.4)	22.3 (16.3 / 6.6)	31.7
	stock, same VRAM	10.5 (4.3 / 6.1)	6.1 (2.0 / 4.1)	6.0 (2.3 / 3.8)	53.7
about 16 GB	engine	4.2 (1.3 / 3.1)	5.1 (2.5 / 2.9)	5.9 (3.1 / 2.8)	86.3
	stock, all experts on CPU	47.6 (17.2 / 27.9)	57.3 (26.6 / 30.8)	61.9 (32.1 / 28.8)	9.5
	stock, same VRAM	23.3 (9.3 / 13.8)	35.0 (20.2 / 15.0)	36.5 (22.8 / 13.7)	19.5
	engine	5.7 (2.3 / 3.4)	13.2 (7.1 / 6.3)	11.6 (6.4 / 4.6)	50.2
	engine, caches emptied before every prompt	7.4 (3.9 / 3.6)	9.5 (6.0 / 3.4)	10.8 (7.4 / 3.3)	73.6

Same-VRAM stock ÷ engine: 1.4 / 1.4 / 1.4 (ample), 2.5 / 1.2 / 1.0 (about 32 GB) and 4.1 / 2.6 / 3.1 (about 16 GB). At about 16 GB with 512-token prompts, the engine's decoding time per session ranged from 2.9 to 9.8 s.

F.6 Generation test (§7)

Table F18: GSM8K by pod (R20). Differences from E (points) with each pod's own paired-bootstrap interval, which was not used for any judgment. Supports §7.4 ("differences across pods") and §7.6.

Pod (GPU, read speed; problems)	Configuration	Accuracy (%)	Difference [pod interval]	Lost/gained	McNemar p	Mass	Mean tokens
Pod A2 (A40, 11.3 GB/s; 0–199)	E	94.5	–	–	–	0	304.9
	Q	93.5	−1.0 [−3.0, +1.0]	3/1	0.625	0.1573	312.3
	S	94.5	0.0 [−2.5, +2.5]	3/3	1.00	0.1688	303.9
	T	92.5	−2.0 [−5.0, +0.5]	6/2	0.289	0.2090	309.0
	C	92.5	−2.0 [−4.5, 0.0]	5/1	0.219	0.2285	317.0
	H	84.5	−10.0 [−15.0, −5.5]	22/2	< 0.001	0.3694	334.1
Pod B (RTX 3090, 2.9 GB/s; 200–399)	E	92.5	–	–	–	0	318.7
	Q	94.5	+2.0 [−1.5, +5.5]	4/8	0.388	0.1580	308.9
	S	91.5	−1.0 [−4.0, +2.0]	6/4	0.754	0.1673	314.2
	T	89.5	−3.0 [−7.0, +0.5]	10/4	0.180	0.2073	323.4
	C	86.0	−6.5 [−11.0, −2.0]	18/5	0.011	0.2275	317.1
	H	87.0	−5.5 [−10.0, −1.0]	17/6	0.035	0.3707	320.9
Pod C (A40, 11.4 GB/s; 400–499)	E	95.0	–	–	–	0	308.6
	Q	95.0	0.0 [−3.0, +3.0]	1/1	1.00	0.1596	306.8
	S	96.0	+1.0 [−2.0, +4.0]	1/2	1.00	0.1685	311.2
	T	96.0	+1.0 [0.0, +3.0]	0/1	1.00	0.2117	310.8
	C	95.0	0.0 [−4.0, +4.0]	2/2	1.00	0.2295	308.8

Pod (GPU, read speed; problems)	Configuration	Accuracy (%)	Difference [pod interval]	Lost/gained	McNemar p	Mass	Mean tokens
	H	84.0	−11.0 [−18.0, −5.0]	11/0	0.001	0.3703	330.7

The lost and gained counts add up to those of Table 9 (for C, 5 + 18 + 2 = 25 lost and 1 + 5 + 2 = 8 gained). HumanEval ran only on pod C (Table 9).

Table F19: Generation length, truncation and first divergence from E (R20, pooled over pods). "Mass" averages all decode layer-tokens of a configuration (§7.1); "mean problem mass" averages the problems' own masses. Supports §7.3.

Task	Configuration	Mean generated tokens	Truncated at 1,024 (%)	First divergence from E, median (tokens)	Mass	Mean problem mass
GSM8K (prompt tokens, min / mean / max: 46 / 83 / 180)	E	311.2	4.2	–	0	0
	Q	309.9	3.0	22	0.158	0.166
	S	309.5	2.6	18	0.168	0.176
	T	315.1	4.4	20	0.209	0.216
	C	315.4	4.2	18	0.228	0.235
	H	328.1	4.8	13	0.370	0.379
	Qr / Sr (problems 0–99)	298.8 / 305.7	2 / 2	identical to Q / S	0.158 / 0.168	0.165 / 0.175
HumanEval (prompt tokens, min / mean / max: 74 / 170 / 427)	E	243.7	2.4	–	0	0
	Q	248.6	2.4	150.5	0.166	0.171
	S	244.6	1.8	152	0.175	0.180
	T	240.0	1.8	150	0.211	0.207
	C	237.0	1.2	150	0.225	0.221
	H	246.0	2.4	138	0.369	0.370

The measured masses were 1.76–2.26 times (GSM8K) and 1.76–2.37 times (HumanEval) the masses predicted for H1 (Q 0.07, S 0.08, T 0.10, C 0.11, H 0.21).

Table F20: Problem-level relations within each configuration (R20; post hoc). Supports §7.4 and §3.3 (condition 2).

Task	Configuration	Logistic slope within the configuration (log-odds per 0.1 of mass)	Answers different from E	Correlation of problem mass with generation length	Mean mass, problems under 200 tokens (n‡)	Mean mass, problems over 600 tokens (n‡)
GSM8K	Q	−7.32	28	−0.71	0.180 (102)	0.128 (33)
	S	−5.59	26	−0.71	0.194 (111)	0.137 (32)
	T	−7.02	36	−0.51	0.219 (107)	0.174 (33)
	C	−4.67	42	−0.49	0.238 (110)	0.195 (33)
	H	−3.11	67	−0.61	0.389 (105)	0.343 (46)
HumanEval	Q	not computed	2	−0.46‡	0.179 (74)‡	0.152 (5)‡
	S	not computed	1	−0.42‡	0.187 (75)‡	0.159 (6)‡

Task	Configuration	Logistic slope within the configuration (log-odds per 0.1 of mass)	Answers different from E	Correlation of problem mass with generation length	Mean mass, problems under 200 tokens (n‡)	Mean mass, problems over 600 tokens (n‡)
	T	not computed	1	+0.26‡	0.200 (78)‡	0.210 (5)‡
	C	not computed	3	+0.25‡	0.212 (76)‡	0.217 (5)‡
	H	not computed	8	−0.10‡	0.371 (74)‡	0.355 (7)‡

The planned H3, pooled over configurations: GSM8K 0.088 [−0.137, +0.321] (2,500 rows, 7.96% with a different answer), HumanEval 0.873 [0.414, 1.499] (820 rows, 1.83%) and both tasks 0.165 [−0.048, +0.393] (3,320 rows, 6.45%). The 15 different HumanEval answers, 8 of them under H, come from 10 problems.

G Experiment B (R21): equal mass, different composition

G.1 Plan and timeline

We wrote the plan of Experiment B in Korean before the run, and an addendum after we had seen the results of both the primary and the auxiliary runs. The addendum does not change any judgment.

Table G1: Timeline (2026-10-05, UTC). All runs used one RTX 3090 community pod.

Time	Event
06:27:44–06:31:43	Stage 1: download of the checkpoint and construction of the session pool; no model computation
about 06:29	Plan written
06:31–06:32	Stage-2 run bundle built, containing the plan
06:33:05	Stage 2 starts; the head of the log records the checksums of the plan and the code
06:33:22–06:37:15	Model computation of the primary run
06:40:58	Auxiliary run bundle built, after the primary result had been seen
06:41–06:48	Auxiliary run (exploratory)
about 06:59	Addendum written

- **Code checksums.** The plan lists the md5 of seven code files. The log printed those of the five files in the run bundle (expb.py, analyze_b.py, trace_next.py, the run script and prep_data.py), and all matched. The auxiliary run script called the expb.py and analyze_b.py in the stage-2 working folder. The md5 lines that this script printed were not kept; that the original-routing stream gave identical NLL, KL and agreement in both runs is indirect evidence.

G.2 Details of the design

- **Subsets and pairs.** For each arm and each scored decode layer-token, the arm's router logits give the top-10 set W and the softmax gates g. All 511 non-empty subsets of the experts of ranks 2–10 (all but the first choice) are formed, and for each subset we compute its mass s, Σg^2 and concentration $c = \Sigma g^2/s^2$. A subset is admissible if $|s - m| \le \delta$. *Among pairs (A, B) of admissible subsets with* $|s(A) - s(B)| \le \varepsilon$ *and* $c(A) > c(B)$*, we take the pair with the largest* $c(A) - c(B)$*, breaking ties by the first pair in bitmask order. The heavy arm substitutes A and the light arm B.* $\delta = 0.2 \cdot m$ and $\varepsilon = 0.01 \cdot m^*$ were fixed before the run. The first choice is never substituted.
- **Why pairs.** The rule we first considered took the admissible subsets with the largest and the smallest Σg^2. When we tested the code on a small random model before fixing the plan, under this rule the heavy arm took subsets near the top of the admissible range, so the masses of the two arms drifted apart systematically. Choosing by concentration instead made the light arm take the top of the range. We therefore fixed a rule in which the two arms take the two members of one pair whose masses differ by at most ε. These tests used no output of the real model.

- **Freed slots.** Filling removes the substituted experts, selects the top 10 of the remaining logits and recomputes the softmax over them (as in §3.1). Renormalization leaves the slots empty and rescales the remaining g to sum to one. Where no pair exists, both variants equal the original routing.
- **Sessions.** We used the sessions with 0-based within-domain indices 4, 6, 8, 10, 12, 14, 16 and 17. These indices avoid the emulation set of R4 (odd within-domain indices 1–15) and the engine set (probably the first four, C.1). We checked the selection of R4 in its code (odd positions within each domain). Index 17 of chat_ko is the only packed session of that domain (the stage-1 log lists 17 single-document sessions and 1 packed session for chat_ko); every math_chat session is packed. The pool was rebuilt on the R21 pod, this time with the data and model revisions logged (G.5). The number of sessions per domain (32, 32, 32, 32, 18, 32) matches the earlier pool, but we did not check on token files that the sessions themselves are the same.
- **Analysis.** $\Delta\text{ppl} = 100 \cdot (\exp((\Sigma\ \text{NLL_arm} - \Sigma\ \text{NLL_original}) / \text{N}) - 1)$, with the sums over all scored tokens of the 48 sessions (N = 73,680). The session-level bootstrap uses one set of 2,000 resamples, `numpy.random.default_rng(15).integers(0, 48, size=(2000, 48))`, shared by every statistic; each resample pools the NLL sums and token counts of the drawn sessions, and 95% intervals are the 2.5th and 97.5th percentiles. The realized mean mass is averaged over all decode layer-tokens, counting layer-tokens without substitution as zero (§3.1).
- **Code checks.** Just before the plan was fixed we checked the following on a small random model (test_expb.py). The expb.py of that time differed from the file used in the run only in the default value and help text of –arms, which the runs set explicitly. After the run, the independent checker reran the tests on the files used in the run and all passed; both outputs are kept with the results.
  - Pair selection matched a direct enumeration of all pairs in all 1,800 cases.
  - Filling and renormalization behaved as defined.
  - The NLL of the original routing matched that computed by transformers.
  - With an unreachable target, the output equaled that of the original routing.
  - The substituted masses lay within the admissible range, and the masses within each pair differed by at most ε.
  - The heavy arm had the larger concentration and substituted fewer experts.
  - Runs on the same input were deterministic.

G.3 Results

Table G2: Results per arm. Brackets give 95% session-bootstrap intervals. The perplexity of the original routing is 3.1370. "Defined" is the fraction of layer-tokens where a pair existed and substitution took place; "admissible" is the fraction with at least one admissible subset. Mass (all) is averaged over all decode layer-tokens; mass, concentration, count and rank (substituted) are averaged over layer-tokens with a substitution. Rank is the rank within W (first choice = 1). Only the primary condition was planned in advance; the others are exploratory runs made after the primary result was known.

Condition	Arm	Δppl (%)	KL	Argmax agreement	Defined	Admissible	Mass (all)	Mass (substituted)	Concentration	Count	Rank
m* 0.10, filling (primary)	heavy	+0.429 [+0.112, +0.768]	0.0116	0.9685	0.334	0.996	0.0352	0.1054	0.798	1.44	5.96
	light	+0.014 [−0.105, +0.134]	0.0064	0.9760	0.334	0.996	0.0352	0.1054	0.484	2.16	8.10
m* 0.15, filling	heavy	+4.355 [+2.756, +6.095]	0.0562	0.9277	0.946	0.997	0.1451	0.1534	0.776	1.50	4.97
	light	+0.494 [+0.189, +0.802]	0.0186	0.9572	0.945	0.997	0.1450	0.1535	0.448	2.40	7.43

Condition	Arm	Δppl (%)	KL	Argmax agreement	Defined	Admissible	Mass (all)	Mass (substituted)	Concentration	Count	Rank
m* 0.10, renormalization	heavy	+0.362 [+0.113, +0.596]	0.0130	0.9658	0.331	0.996	0.0349	0.1054	0.798	1.44	5.96
	light	−0.029 [−0.295, +0.201]	0.0091	0.9726	0.330	0.996	0.0348	0.1055	0.485	2.15	8.11
m* 0.05, filling	heavy	+0.006 [−0.048, +0.061]	0.0013	0.9886	0.009	0.638	0.0005	0.0520	0.789	1.49	6.20
	light	−0.061 [−0.111, −0.015]	0.0013	0.9885	0.009	0.638	0.0005	0.0521	0.468	2.31	8.14

Table G3: Comparison of the arms. R = Δppl_heavy / Δppl_light, and R_m is the ratio of each arm's Δppl divided by its realized mean mass. The difference is Δppl_heavy − Δppl_light (points). "Rule output" applies the planned decision order as written; only the output for the primary condition is a planned judgment.

Condition	R	R_m	Difference	KL ratio	Mass difference	Rule output
m* 0.10, filling (primary)	30.4 [−87.0, 100.9]	30.4	+0.415 [+0.128, +0.702]	1.826 [1.726, 1.927]	−0.000004	no judgment (interval of the light arm includes zero)
m* 0.15, filling	8.82 [5.99, 16.94]	8.82	+3.861 [+2.436, +5.404]	3.024 [2.876, 3.186]	+0.00009	heavy substitutions are more harmful
m* 0.10, renormalization	−12.4 [−41.6, 38.4]	−12.4	+0.391 [+0.021, +0.778]	1.437 [1.354, 1.521]	+0.00012	no judgment (interval of the light arm includes zero)
m* 0.05, filling	−0.094 [−2.068, 0.862]	−0.095	+0.067 [+0.007, +0.130]	1.067 [1.030, 1.106]	−0.000004	"direction only: many light substitutions are more harmful" (the defect in G.4)

Table G4: Δppl (%) by domain, 8 sessions per domain, without intervals.

Condition	Arm	ko_wiki	en_wiki	python_code	chat_en	chat_ko	math_chat
m* 0.10, filling (primary)	heavy	+1.05	+0.98	+0.74	−0.36	+0.95	−0.77
	light	+0.10	−0.05	+0.19	−0.17	+0.20	−0.18
m* 0.15, filling	heavy	+8.29	+6.64	+5.08	+1.50	+7.59	−2.53
	light	+1.23	+0.85	+0.79	−0.02	+0.92	−0.80
m* 0.10, renormalization	heavy	+0.82	+0.44	+0.62	+0.14	+0.75	−0.58
	light	−0.04	−0.74	−0.16	+0.17	+0.34	+0.26
m* 0.05, filling	heavy	+0.16	−0.09	+0.00	+0.04	+0.00	−0.08
	light	+0.01	−0.11	−0.10	−0.06	−0.03	−0.08

- The heavy arm lowered perplexity on math_chat in all four conditions (only −0.08 at m* 0.05). We did not investigate why.
- The next-token accuracy of the original routing (argmax equal to the target token) was 0.7406; at m* 0.15 it was 0.7326 for the heavy arm and 0.7399 for the light arm.

G.4 Departures from the plan and other notes

- **A defect in the decision rule.** The planned decision order implicitly assumed a positive Δppl for the light arm. If the whole interval for the light arm's Δppl lies below zero, the first step is passed, R is negative, and later steps can output "many light substitutions are more harmful". This happened at m* 0.05. There the heavy arm was worse (difference +0.067 [+0.007, +0.130]), so the rule's output is the opposite of what the data show. The primary condition ended at the first step and was not affected.
- **Realized mass.** The plan stated in advance that the realized mean mass falls below m* when many layer-tokens have no pair. We did not, however, compute this fraction from the model's logits before the run, because the step of the first design that would have chosen δ from the logits of the original routing was dropped from the plan. In the primary condition a pair existed in 33.4% of layer-tokens, and the realized mean mass was about a third of the target (0.035).
- **Auxiliary runs.** The plan excluded m* 0.05, m* 0.15 and renormalization and stated that, if run later, they would be reported as exploratory. We ran the three conditions after seeing the primary result, on the same pod with the same code. We made no correction for multiple comparisons.
- **Recovery of raw data.** The raw-data lines of the auxiliary run (per-session NLL and other values, 20,084 compressed bytes) and the revision lines of the stage-1 log were not copied to files at first. Before the pod was deleted, we recovered them with a script from the work record that had read the pod log. We checked the raw data with the md5 of every line and of the final line (compressed 68444edc…, raw d01cdeea…), and the analysis rerun on the recovered data reproduced the pod's output.
- **Other limits.** The limits stated in advance remain: weight and rank are confounded; every layer-token receives the same target mass; the streams of the two arms diverge after earlier substitutions; and the measurement uses one model in BF16 emulation under teacher forcing.

G.5 Environment

- **Pod.** One RTX 3090 (24 GB) community pod with 32 vCPUs by specification (128 CPUs were visible inside the container). The checkpoint was on a 200 GB persistent volume; the model folder measured 152G (du) after the download. The container memory limit was not recorded. Stage 1 took about 4 minutes (223 s of download), the model computation of the primary run 233 s and that of the auxiliary run 383 s.
- **Software.** torch 2.8.0+cu128 and transformers 4.57.6. The fast kernels for linear attention were not installed, so the torch implementation was used.
- **Revisions.** Model: Qwen/Qwen3-Next-80B-A3B-Instruct 9c7f2fbe84465e40164a94cc16cd30b6999b0cc7. Data: wikimedia/wikipedia b04c8d1ceb2f5cd4588862100d08de323dccfbaa, codeparrot/codeparrot-clean-valid 4db92d2ec0c1b4c41eeb439cfae16854511d9dcd, HuggingFaceH4/ultrachat_200k 8049631c405ae6576f93f445c6b8166f76f5505a, maywell/koVast b0704da17669a3f32b5f11bb11c5b27620d27677, AI-MO/NuminaMath-CoT 9d8d210c9f6a36c8f3cd84045668c9b7800ef517.
- **Inputs.** The md5 of the session pool (178 × 2,048 tokens) is 5190562771148a2112c701aac2e9cdd8, and that of the 48 selected sessions is 896cb46ca22ab9377a3be20fa4002dab.

H Full versions of passages shortened in Sections 6 and 7

Sections 6 and 7 give shortened versions of the following passages. They are reproduced here in full, as written before shortening; table and section references are to the main text.

H.1 Choosing two policies (§6.1)

How we chose. We chose the policies in three steps. First, we compared components with the router-logit simulator (R16). With a 25% cache, when τ was varied so that the simulator's estimated time per token (including contention with background reads) equaled that of the 0.15 rule, background admission ($\beta = 0.5$)

had 22–34% less mass, and refilling the resident set with the most frequent experts every 256 tokens had 9–18% less. Policies that substitute every miss without synchronous fetches had the shortest estimated times but more mass than the 0.15 rule (1–82% more for an approximation of SwapMoE that replaces the resident set every 16–256 tokens, and 114–387% more without replacement). Second, we measured candidates in the engine on 6 sessions and 3 topic-change sequences (R17). Third, we measured them again on 24 sessions (R18) and chose two policies from those results: - quality-first: the 0.15 rule + frequency-based eviction + background admission ($\beta = 0.5$); - speed-first: the first-choice rule + frequency-based eviction + background admission ($\beta = 0.5$).

On the 6 sessions, the quality-first candidate was the 0.15 rule with frequency-based eviction (plus background admission with $\beta = 1$ when topics change often). On the 24 sessions, adding background admission with $\beta = 0.5$ gave the smallest perplexity increase, so we changed the candidate. The values from R18 were therefore measured on the data used for the choice. The checks after the choice are the 24 topic-change sequences (R19) and the planned generation test (§7). The topic-change sequences, however, concatenate the same 24 sessions as R18 in new combinations, so only the test in §7 uses entirely new data.

A conclusion that changed with the sample. In the 3 topic-change sequences each measured before the choice, speed-first had higher perplexity than the 0.15 rule (R17: +0.36% [−0.55, +1.35]; R18: +0.69% [−0.06, +1.48]). With 24 sequences the difference was −0.30%. The per-sequence values of the 24 sequences ranged from −2.32% to +1.96%; among all combinations of 3 of them, about 11% had a mean of +0.36% or more and about 3% a mean of +0.69% or more. The 3-sequence values are thus largely within this variation, although the R18 value lies in the tail. The three sets combine sessions differently, so this comparison is approximate.

Speed across hosts. For the same 6 sessions, time per token was 24–41% longer on the R18 host than on the R17 host. SSD reads were similar on the two hosts (5.8 and 6.1 GB/s), whereas they had 6 and 32 vCPUs. The CPU may have been short for handling misses, but we did not verify this. The time differences between configurations also varied with host and sample. Speed-first minus the 0.15 rule was −4.8% (R17, 6 sessions), −7.3% (R18, 24 sessions), −8.3% (R18, the 6 sessions of R17), −4.9% (R18, 3 topic-change sequences) and −4.4% (R19); quality-first minus the 0.15 rule was +1.9% (R17), +0.4% (R18) and +1.3% (R19). Rerunning a configuration on the same host changed its time per token by 2–3%, while perplexity stayed within 0.04% (R17). We therefore report ratios and time differences only for the hosts measured and do not claim that they hold on other machines.

Speed in greedy generation (continued from Table 7). In all four measurements, speed-first took 3.8–9.4% less time per token than the 0.15 rule, and quality-first was within −1.3% to +1.3%. These differences, however, include run-to-run variation. The configurations of a chunk ran one after another rather than interleaved, and when Q and S were rerun on the same 100 problems on the same pod (A2), the outputs were identical token for token, yet the time per token was 3.6% and 9.5% shorter than in the original runs. This variation is of a similar size to the differences in Table 7. The four configurations with a 25% cache generated 64–74 tokens per second, and the first-choice rule with a 12.5% cache (H) took 14.5–28.9% longer per token than the 0.15 rule. In this test each problem started from an empty cache and generations were short (237–328 tokens on average per configuration), so the conditions differ from those of §6.2. Fetching every miss read experts through the page-cache-first path in this test, so we do not report its speed.

Components left out of the two policies. We also measured a component that gives each layer a different number of slots. The engine used one fixed allocation for all sessions: the average of the per-layer allocations computed from the miss curves of the prompts of the 24 sessions, so the allocation was derived from the prompts of the sessions measured. On the 24 sessions of R18, adding it gave 3.8% (0.15 rule) and 6.8% (speed-first) less mass and perplexity changes of −0.11% [−0.38, +0.17] and −0.26% [−0.53, −0.00]; its time could only be compared across separate runs. Our records assessed its gains in quality and speed as small (R18), and we did not include it in the two policies. Frequency-based eviction alone also had a small effect (with the 0.15 rule: −0.26% [−0.53, +0.03] within a topic and −0.10% [−0.29, +0.08] across topic changes). On the 6 sessions measured first (R17) the within-topic effect was −0.58% [−1.09, −0.16], with an interval below zero, and the 6-session quality-first candidate rested on it; on 24 sessions the effect

was smaller and its interval included zero. Background admission evicts the resident expert with the lowest frequency score, so both policies use frequency-based eviction together with it.

H.2 Comparison with stock llama.cpp (§6.2)

Conditions. We compared the engine using the 0.15 rule with stock llama.cpp on one machine (R13): an RTX 3090 (24 GB), an NVMe SSD (Fanxiang S880, PCIe 4.0, 5.7 GB/s O_DIRECT reads) and 16 threads (32 vCPUs on the pod), with a container RAM limit of 57.7 GiB. The model file is 45.2 GiB. The about-32 GB and about-16 GB RAM conditions were created by other processes holding 25 and 41 GiB (leaving about 32.7 and 16.7 GiB). The engine used the 0.15 rule with a 25% cache and 15.7 GiB of VRAM. We measured two ways of reading missed experts: always from the SSD (SSD only), or copying an expert from RAM when it is in the page cache (page cache first, the current default). With ample RAM we did not measure page cache first separately and use the run in which an automatic switch, based on the share of expert pages in the page cache, turned this path on. Stock llama.cpp kept the experts of 14 of the 48 layers on the GPU to match our VRAM (15.8 GiB) or kept all experts on the CPU (2.8 GiB). For prompts of 128, 512 and 1,536 tokens we processed 6 sessions, one per domain, and then decoded the next 256 tokens of the data one at a time. Time to answer is the time to the first token plus the time to decode 256 tokens, reported as the median over sessions. Stock llama.cpp at about 32 and 16 GB was measured on the first 3 sessions only. Computing the engine's values on the same 3 sessions changed the ratios by less than 3% with page cache first and with the automatic switch, and by at most 3.2% with SSD only.

Results. The gap between the engine and the same-VRAM stock configuration was largest at about 16 GB (Table 8, Figure 3). In this condition, the same-VRAM stock configuration took 3.4–4.1× (page cache first) and 3.0–3.6× (SSD only) as long as the engine to answer, and the engine decoded 89.3–94.5 tokens/s against 31.8. Here the automatic switch chose the SSD-only path, with 89.5 tokens/s and ratios of 2.9–3.4. Its times, however, exclude the residency check at the start of prompt processing; if that check costs as much as the one measured at generation setup (about 0.24 s), the ratios are about 2.8–3.3. At about 32 GB, the ratios with page cache first were smaller (1.8 / 1.2 / 1.2), and for 512- and 1,536-token prompts the stock configuration reached the first token slightly sooner (1.5 / 2.0 s vs. 1.6 / 2.2 s). In this condition the stock configuration processed the 128-token prompts first and was slow on all three sessions (first token 2.1–3.0 s), apparently while filling the page cache, so the 128-token ratios (1.8, and 1.4 for SSD only) may favor the engine. With ample RAM, page cache first gave a ratio of 1.4. In contrast, the SSD-only path took 6–17% longer to answer than the stock configuration with ample RAM, and 7–12% longer for 512- and 1,536-token prompts at about 32 GB (4% for 512-token prompts on the 3 sessions measured for stock). In these two conditions, the engine answered sooner at every prompt length only with the page-cache-first path. In decoding speed alone, the engine was faster in every condition (1.2–1.5× with ample RAM and 1.4–1.5× at about 32 GB).

Page cache and loading. At about 16 GB, page cache first answered 11–13% sooner than SSD only, and in this condition the automatic switch measured 25–32% of the expert pages in the page cache. On another host with the same SSD model, 34.7% of the model file remained in the page cache after a run of the engine alone, even when the page cache was emptied before the run, and page cache first answered in 4.0 / 4.7 / 5.4 s (R14). Because llama.cpp's loader reads the whole file when it loads the model, these pages appear to have been filled by the engine's own loading. On a real 16 GB PC, the operating system and other programs use RAM, so less may remain cached and the gain may be smaller. When the expert cache and the page cache were emptied before every prompt, the two paths differed by about 1% (R14; Table F16b). Time to answer excludes loading the model, which took 17.2–20.3 s for the engine and 19.6–24.1 s for stock llama.cpp at about 16 GB.

Quality. In the same measurement the engine's perplexity was 5.5405, 1.6% higher than stock llama.cpp's (5.4526 with the same VRAM, 5.4538 with all experts on the CPU), over 4,608 tokens (6 sessions × 3 prompt lengths × 256 tokens). When the engine fetches every miss, its output matched stock llama.cpp bit for bit on a small model, and on the real model its perplexity differed by 0.01–0.04%, the same order as the difference between stock GPU and CPU configurations (the R7 engine, on an A40). The 1.6% is much larger than that difference.

Another machine. On another machine with slower disk reads (XG7000, 4.3 GB/s O_DIRECT reads; R12), the absolute values differed substantially. At about 16 GB, the engine (page cache first) answered in 5.7 / 13.2 / 11.6 s and the same-VRAM stock configuration in 23.3 / 35.0 / 36.5 s (4.1 / 2.6 / 3.1×), and the engine's decoding time per session varied widely, from 2.9 to 9.8 s with 512-token prompts. On that machine, an engine run that emptied the caches before every prompt decoded at 73.6 tokens/s, faster than this run (50.2 tokens/s), but the cache states differed, so the comparison was not controlled. We did not separate the causes of the difference between the two machines.

H.3 The generation test (§7)

Design.

Tasks. We used the first 500 problems of the 1,319-problem GSM8K test set in a seed-14 shuffle and all 164 HumanEval problems. Each prompt is a single user message in Qwen's chat template without a system prompt. For GSM8K, the problem is followed by an instruction to reason step by step and put the final answer within $\square$. For HumanEval, the prompt gives the function signature and docstring and asks for the complete function in a single Python code block.

Generation. Greedy decoding generates up to 1,024 tokens and stops at the end-of-generation token. Each problem starts from an empty cache: the prompt is computed exactly, its routing fills the cache, and generation follows. We empty the cache because, in a preliminary check, the slot layout left by the previous problem changed the tie-breaking of eviction and made a problem's output depend on the previous problem. The engine is that of §6.1 with a greedy generation mode added; cache behavior is unchanged.

Configurations. There are six configurations. The reference E fetches every miss. Q (quality-first), S (speed-first), T (the 0.15 rule alone) and C (the first-choice rule alone) use a 25% cache, and H is the first-choice rule with a 12.5% cache (64 slots per layer). During generation, all configurations except E read experts from the model file with O_DIRECT (the pods' storage was not recorded; Appendix C.5). E used the page-cache-first path; because E waits for every miss, its output does not depend on where experts are read from, but we do not report its speed.

Runs. Following a change to the plan made before generation, three pods split the problems: A2 (A40) took GSM8K problems 0–199, B (RTX 3090) 200–399, and C (A40) 400–499 and all of HumanEval. The six configurations of a chunk always ran on the same pod, so every paired comparison is within one pod. The three pods used an experiment bundle with identical md5. At the end, Q and S were run again on GSM8K problems 0–99 (Qr, Sr) to measure run-to-run differences arising from the arrival times of background reads. Before the main run, six pods stopped while building the engine or earlier, and none of them started generating.

Scoring and measures. A GSM8K answer is correct if the first number in the last $\square$ (or, without one, the last number in the response) equals the reference answer. HumanEval was scored with the rule as implemented: the last closed code block opened by ```` ```python ```` (or ```` ```py ````, ```` ```Python ```` or an untagged ```` ``` ````) is used; without such a block, everything after the last ```` ```python ````; without that, the whole response. With the prompt's import lines prepended, the code must pass the official tests within 10 s. If the code lacks a definition of the function, it is run after the prompt. The wording of the plan specified the last Python code block of the response, or everything after the last ```` ```python ```` if the block is unclosed; it differs from the implementation when a closed block is followed by an unclosed one (§7.5). Agreement with E is measured in two ways: final-answer agreement is the share of problems whose extracted number (GSM8K) or pass/fail outcome (HumanEval) equals E's, and token identity is the share whose generated token sequence is identical to E's. A problem's mass is the average over the layer-tokens generated for it, and a configuration's mass averages over the layer-tokens of all its problems.

Statistics. For accuracy differences (configuration − E) we use 95% percentile intervals from a problem-level paired bootstrap (2,000 resamples, seed 14) and exact McNemar tests. For agreement rates we use Wilson 95% intervals (Appendix D.4, Table D1). The non-inferiority margin is −2 points.

Results.

On GSM8K, accuracy differences followed the order of mass. From the configuration with the least mass to the one with the most, the differences were +0.4 (Q), −0.2 (S), −1.8 (T), −3.4 (C) and −8.4 points (H) (Table 9; Figure 1c). The intervals of Q and S included zero, the interval of T had an upper bound of zero ($p = 0.093$), and those of C and H were below zero. Final-answer agreement also decreased in the order of mass, except that Q and S were reversed (94.4% vs. 94.8%). The line fitted to the five configurations had a slope of −0.41 points per 0.01 of mass, and −0.51 without H. Almost all outputs diverged from E's (token identity 0.4–3.4%; median first divergence at 13–22 tokens), yet 87–95% of the final answers were the same.

On HumanEval, the differences were small and the agreement prediction (H2) failed. The accuracy differences of the five configurations ranged from −2.44 to +0.61 points (Figure 1d); one problem is 0.61 points. Outputs diverged from E's late (median first divergence at 138–152 tokens), and 22–39% were identical token for token. Final-answer agreement was concentrated at 95.1–99.4%. The Spearman correlations of mass with token identity and with final-answer agreement were −0.60 and −0.67. The fitted slope was negative (−0.09 points), but it became +0.16 points without H, so it rests on H alone.

The problem-level prediction (H3) failed. The slope with both tasks pooled was 0.165 log-odds per 0.1 of mass [−0.048, +0.393]. The wording of the plan pools the five configurations for H3 without saying whether to split by task, and we judged it on both tasks pooled (§7.6). The positive slope on HumanEval alone (0.873 [0.414, 1.499]) rests on 15 cases (configuration × problem, from 10 problems) whose answers differed from E's, 8 of them under H.

The predicted absolute masses were wrong. The order of mass (H1) held on both tasks, but the values were 1.8–2.4× the predictions. The predictions came from R18, which decoded 1,535 tokens after a 512-token prompt, whereas in this test each problem started from an empty cache and generated 237–328 tokens on average per configuration after a short prompt (83 and 170 tokens on average). A larger share of tokens generated before the cache filled may explain the higher values, but we did not verify this.

The two chosen policies. Quality-first was non-inferior on GSM8K (lower bound −1.4) but not shown to be non-inferior on HumanEval (lower bound −3.05; 2 problems lost, 0 gained). Speed-first was non-inferior on both tasks by the planned rule (lower bounds −2.0 and −1.84), but its GSM8K lower bound equals the margin. The point estimates of both policies were within the H4 range (−1.5 to +0.5 points) on both tasks; the first part of H4 failed because T and C on GSM8K and T on HumanEval were outside it.

Repeated runs. Qr and Sr were identical token for token to Q and S on all 100 problems. They ran on the same pod at the end of its run, about two hours after Q and S, so we did not measure whether this holds across pods or disk speeds. Their time per token was 3.6% and 9.5% shorter (§6.1).

Post-hoc analyses [exploratory].

Problem-level relations within a configuration. On GSM8K, the logistic slope within each of the five configurations was negative (per 0.1 of mass: Q −7.3, S −5.6, T −7.0, C −4.7, H −3.1). On GSM8K, a problem's mass was negatively correlated with its generation length (−0.49 to −0.71 across configurations); under S, for example, problems that ended within 200 tokens had a mean mass of 0.194 and those longer than 600 tokens 0.137. Longer generations appear to have less mass per layer-token and to diverge from E more easily, pulling the problem-level slope in the opposite direction. On HumanEval, the sign of this correlation varied across configurations (−0.46 to +0.26, computed from the per-problem records). We therefore do not use a problem's mean mass in this form as a problem-level measure. We did not run analyses that compare configurations within the same problem or control for generation length.

Sensitivity of the non-inferiority judgments. Recomputing the planned intervals (paired bootstrap percentiles) with 20 bootstrap seeds gave lower bounds of −1.4 to −1.2 for Q and −2.0 to −1.8 for S on GSM8K. On HumanEval, Q's lower bound was always −3.05, and S's was −1.83 for 19 seeds and −2.44 for one (seed 16). S lost only one problem on HumanEval, so the exact 2.5% quantile of the bootstrap distribution is −1.83 (the probability that this problem is drawn four or more times is 0.019). With seed 16, however, the lower bound fell below the margin, so the judgment can change, rarely, with the seed. In the planned computation (seed 14), exactly 50 of the 2,000 resamples were at −2.44 or below, and the −1.84 in Table 9 is interpolated between −2.44 and −1.83. With Newcombe intervals for paired differences in proportions (Wilson-based), S's lower bounds were −2.04 to −2.08 on GSM8K and −3.05 to −3.32 on

HumanEval, below the margin (the values depend slightly on the implementation; Table D2), whereas Q's GSM8K lower bound stayed above the margin at about −1.4. We therefore regard S's non-inferiority as weak evidence that holds only under the planned interval method.

Differences across pods. The GSM8K loss of C was concentrated on the problems assigned to pod B (A2 −2.0 points, B −6.5, pod C 0.0); on the same pods S had 0.0, −1.0 and +1.0. Pods and problems changed together, so the two cannot be separated. No test comparing C directly with S was planned.

Examples of errors. E solved and S did not solve 10 GSM8K problems. Of the 7 printed by the pods (up to the first three per pod by problem number), 4 had the same final expression as E but a wrong sum, and in 3 S did not finish within 1,024 tokens, so the last number was extracted as the answer. In one of these (problem 271), E was also truncated at 1,024 tokens, and its last number happened to be correct. With only 7 examples we do not generalize.

The HumanEval scoring defect and regeneration.

Our HumanEval scorer extracted the last fenced code block with a regular expression that also accepted untagged fences as opening fences, so a non-Python block preceding the solution could have caused the wrong span to be extracted. We found this defect by reviewing the code while generation was still running and before the pods' records were pooled, but the generated outputs were not retained when the pods were released. After the original results were known, we wrote an addendum to the plan that fixed a reproduction criterion and two corrected extraction rules before any regeneration (Appendix E.4). We then regenerated all 984 HumanEval outputs (164 problems × 6 configurations) from the same engine source, build script and task file on a different host with the same GPU type (A40). All 984 records matched the original run on the six recorded fields: generation length, termination, mass, pass/fail, final-outcome agreement with E and token identity with E. The original token sequences were not available for direct comparison. Re-scoring the regenerated outputs with the two corrected rules, which parse fences line by line, changed no outcome. Under the original rule and both corrected rules, all HumanEval results and planned judgments are therefore the same. (Exploratory) No extracted program changed either, and no response contained a block tagged with a non-Python language. One response under H (HumanEval/76), however, passes because the implemented rule uses the last closed block; a literal reading of the wording of the plan would use its truncated, unclosed final block and fail it. That would lower H's HumanEval accuracy from 91.5% to 90.9% and change no judgment (Appendix E.5, item 2). (Exploratory) The 67 failures under the original rule were 16 truncations at 1,024 tokens, 50 assertion failures in the official tests and 1 syntax error in which the model omitted a closing parenthesis. Of the 20 records truncated at 1,024 tokens, 4 passed. In HumanEval/32, E and T had a single unclosed block, and the truncated code extracted by the rule (everything after the last ```` ```python ````) passed the tests; Q, C and H failed this problem, which is one of the two HumanEval problems Q lost.

I Claims, evidence and limits in detail

This appendix gives each claim of Section 1 with its conditions, evidence, limits and the section that reports it.

Terms - **The 0.15 rule** synchronously fetches a missed expert if it is the router's first choice or its normalized gate weight is at least 0.15, and substitutes the rest. - **The first-choice rule** synchronously fetches only missed first choices. - **Mass (substituted gate mass)** is the sum of the normalized gate weights of substituted experts, averaged over layer-tokens. - **Fetching every miss** is the reference configuration without substitution.

Claim 1. In the emulation, most of the quality loss from substitution appears to come from a few substitutions of heavily weighted experts. [exploratory · emulation · two models, weaker evidence from a third] - Conditions - Tool: a PyTorch emulation that computes model outputs exactly but tracks the cache only as bookkeeping. A missed expert is assumed to be resident from the next token (zero fetch latency). - Models: Qwen3-Next-80B-A3B (BF16; top-10 of 512 experts per layer plus one always-on shared expert), DeepSeek-V4-Flash (its FP8 weights, FP4 experts and FP8 activations reproduced; top-6 of 256; the three hash-routed layers always computed exactly), and gpt-oss-120b (MXFP4 experts; top-4 of 128). - Cache: 25% of the experts in each layer, least-recently-used replacement. - Substitution: the resident

expert with the next-highest router score fills the empty slot, and the gate weights are renormalized over the new selection. - Data: 48 sessions, 8 from each of six domains (Korean and English Wikipedia, Python code, English and Korean chat, math solutions). In each session the first 512 tokens use the original routing, and the remaining tokens (1,535 per session, about 74k in total) are scored by teacher-forced perplexity. - Evidence - Qwen3-Next: the 0.15 rule synchronously fetched 0.8% of expert calls. Its perplexity increase was 0.35%, against 5.67% when every miss was substituted (KL 0.013 vs. 0.085), whereas mass fell by 26%, from 0.076 to 0.056. - DeepSeek-V4-Flash: +4.97% when every miss was substituted, +1.29% with the first-choice rule and +0.20% with the 0.15 rule; mass fell from 0.132 to 0.072 between the first and the last. - gpt-oss-120b (code-domain perplexity): +9.7%, +1.19% and −0.03%. Its 0.15 rule, however, substituted only 1.65% of calls, against 15.2% when every miss was substituted, so it offers weaker evidence that the loss comes from few substitutions. - In both models, fetching only first choices left a loss, so heavily weighted substitutions other than first choices also contribute. - Limits - The emulation is idealized. The same 0.15 rule gave +1.21% in the engine (Q4_K_M, real fetches). - gpt-oss is a chat-format model, so we report its code-domain values only. - These comparisons cannot tell whether gate weight itself is the cause. In Experiment B (Claim 7), which varied only the composition at equal mass, heavy substitutions also lost more, but weight and rank cannot be separated there either, because weight order and rank order coincide within a token. - Relation to prior work: this agrees with the report that replacing the first- or second-ranked expert degraded performance while lower-ranked replacements cost little in MoEs with 60–64 experts per layer (Skliar et al., 2025). In a benchmark with a small cache, however, dropping low-ranked missed experts still lowered accuracy (Hoang et al., 2026). What we add are measurements of substitution loss in three models with 128–512 experts per layer: the share of the loss from heavily weighted substitutions other than the first choice was measured separately in the two models with 128 and 256 experts, and in Qwen3-Next, with 512, we compared the 0.15 rule with substituting every miss. - Section: §5.1

Claim 2. Among policies that always synchronously fetch the first choice, compared on the same inputs, policies with more mass had larger perplexity increases. Mass did not, however, explain differences in loss between inputs. [exploratory · engine · partly planned] - Conditions - Engine: our llama.cpp-based engine running Qwen3-Next-80B-A3B Q4_K_M with 25% of experts cached (128 slots per layer, 10.9 GiB), reading experts from the model file with O_DIRECT. Quality was measured by teacher-forced decoding and compared token by token with fetching every miss in the same run. - Policies: seven (single topic) and six (topic change) combinations of the 0.15 rule or the first-choice rule with frequency-based eviction and background admission. All of them fetch the first choice. - Data 1 (single topic): 24 sessions, four per domain, each with a 512-token prompt and 1,535 scored tokens; RTX 3090 host 1. - Data 2 (topic change): 24 sequences, each concatenating four of the same 24 sessions, with a 512-token prompt and 4,096 scored tokens; the topic changes at scored tokens 1,536 and 3,584; RTX 3090 host 2. - Evidence within the same inputs - Policy-level regression: each 0.01 of mass came with a 0.25-point (seven policies; r = 0.92, 95% interval 0.53–0.99) and a 0.26-point (six policies; r = 0.91, 0.37–0.99) larger perplexity increase. Dropping any one policy left slopes of 0.21–0.27 and 0.22–0.30. - Fixed effects within input groups: 0.23 points [0.20, 0.31] (4 groups) and 0.26 points [0.20, 0.32] (8 groups). - Ranks: in all 12 input groups, the Spearman correlation between the mass rank and the perplexity rank of the policies was 0.79–1.00. - Measurement noise: the same session gave the same perplexity to four decimal places on different hosts. - Planned generation test (Claim 6): GSM8K accuracy differences followed the order of mass (−0.41 points per 0.01; −0.51 without the half-cache configuration). HumanEval did not (Spearman correlations of mass with token identity −0.60 and with final-outcome agreement −0.67, against a threshold of −0.9). - Evidence between inputs - The correlation between the mean mass and the mean loss of input groups was −0.21 (4 groups) and −0.32 (8 groups); within each policy it ranged from −0.44 to −0.02. - The planned problem-level prediction also failed (both tasks pooled: 0.165 log-odds per 0.1 of mass [−0.048, +0.393]). On GSM8K, the slope within a configuration was negative, and a problem's mass was negatively correlated with its generation length (−0.49 to −0.71). - Limits - The scope condition is necessary. With policies that also substitute first choices, mass differed by 26% while loss differed 16-fold in the emulation (Claim 1). - The order did not always hold within the scope (§3.4). With similar or smaller mass, first-choice-rule policies had larger perplexity increases than 0.15-rule policies (speed-first in R18, the 12.5% cache in R17 and the DeepSeek-V4-Flash emulation), and the 0.15 rule with frequency-based eviction and a threshold of 1 had 14% less mass but the same perplexity. The composition of substitutions may therefore affect the loss. In

Experiment B, which varied only the composition at equal mass, heavy substitutions lost more (Claim 7; no judgment under its planned rule). The mass hypothesis concerns the order of policies, and this result suggests that mass does not capture the effect of composition. - With six or seven policies the correlation intervals are wide, and the between-input analysis uses only 4 and 8 groups. - The result was sensitive to sample size. With 6 sessions (R17) the correlation was 0.77, and we first concluded that mass was a poor proxy; 24 sessions reversed that conclusion. - Section: §5.2

Claim 3. A rule that admits experts in the background according to their accumulated substituted mass (threshold 0.5) lowered perplexity, with the largest difference right after topic changes. [exploratory · engine] - Conditions - Rule: each time an expert is substituted, its normalized gate weight is added to a sum that decays with a 512-token half-life. When the sum reaches a threshold, the expert is read in the background and replaces the resident expert with the lowest frequency score, and its sum is reset. At most 8 admissions can be pending per layer. - Comparison: the same policy without the rule, paired token by token (both with frequency-based eviction). - Engine and data: as in Claim 2. We tested thresholds of 0.5 and 1. - Intervals: 95% intervals from 2,000 resamples of 128-token blocks (each within one session) drawn from the pooled set of a run's blocks; for topic changes, also sequence-level intervals and sign tests. - Evidence - Single topic, 24 sessions: −0.28% [−0.53, −0.04] with the 0.15 rule and −0.54% [−0.84, −0.26] with the first-choice rule. - Topic change, 24 sequences: −0.69% and −1.51%. Each was lower in 20 of 24 sequences (sign test $p = 0.002$), and the sequence-level intervals excluded zero. - The 512 tokens after each change: −1.34% and −3.53%. These windows hold 25% of the tokens but about half of the loss. - Limits - A threshold of 1, tried with the 0.15 rule, made no difference within a topic (+0.00% [−0.19, +0.19]); it was not measured on the 24 topic-change sequences (R19). - Topics change in the middle of the text decoded under teacher forcing rather than with a new prompt, which is likely harsher than a new topic arriving with a user prompt (not measured). - The topic-change sequences reuse the single-topic sessions, so the two data sets are not fully independent. - With 6 sessions (R17), the single-topic effect of a threshold of 0.5 with the 0.15 rule was not significant (+0.30%). - Section: §5.3

Claim 4. On the two hosts we measured, two policies chosen from these results were 2.5–3.0× faster per token than fetching every miss, with 0.7–1.5% higher perplexity. [chosen from the results · two hosts · one run per configuration] - Conditions - Policies: quality-first is the 0.15 rule plus frequency-based eviction (EMA 1/128, counting only experts the router asked for) and background admission (threshold 0.5). Speed-first is the first-choice rule plus the same two components. - Selection: chosen from a simulation (R16) and engine measurements (R17, 6 sessions; R18, 24 sessions). The checks after selection are topic-change sequences that recombine the same 24 sessions as R18 (R19) and the planned generation test (R20). - Measurement: the engine and data of Claim 2 with 25% cache. Time is the mean per-token time of teacher-forced decoding. Each configuration ran once; rerunning a configuration changed its time by 2–3% (R17). - Hosts: R18, RTX 3090 with 6 vCPUs (16 threads) and a 5.8 GB/s SSD. R19, RTX 3090 with 16 vCPUs (13 threads) and a 5.8 GB/s SSD. - Evidence - Against fetching every miss (R18 / R19): - quality-first: 2.80× / 2.48× faster, perplexity +0.67% / +0.97% - speed-first: 3.03× / 2.62× faster, perplexity +1.00% / +1.46% - Against the current engine with the 0.15 rule alone: - speed-first: perplexity −0.20% [−0.54, +0.13] / −0.30% [−0.57, −0.03] (sequence-level [−0.68, +0.09]), with 7.3% / 4.4% less time. We therefore claim only that it is not worse, not that it is better. - quality-first: perplexity −0.54% [−0.83, −0.25] / −0.78% [−1.03, −0.54], with time +0.4% [−0.5, +1.5] / +1.3% [+0.4, +2.2]. - Greedy generation (R20, four measurements on three pods): speed-first took 3.8–9.4% less time per token than the 0.15 rule, and quality-first was within −1.3% to +1.3% of it. Rerunning quality-first and speed-first on the same 100 problems on the same pod, however, gave identical outputs but 3.6% and 9.5% less time per token, so these differences include run-to-run variation. - Limits - Some values come from the data used to choose the policies (R18). - For the same 6 sessions, time per token was 24–41% longer on a different host, and the time difference between policies also varied by host and sample (speed-first minus the 0.15 rule: −4.8%, −7.3%, −8.3%, −4.9%, −4.4%). We report ratios only for the hosts measured. - The reference, fetching every miss, is the slowest policy. The comparison with stock llama.cpp is Claim 5. - Section: §6.1

Claim 5. On one consumer GPU, the engine with the 0.15 rule and its default read path (page cache first) answered faster than stock llama.cpp configured to use the same VRAM, and the gap was largest at about 16 GB of RAM. [one machine · 0.15 rule] - Conditions - Machine: RTX 3090

24 GB, NVMe SSD (PCIe 4.0, 5.7 GB/s with O_DIRECT), 16 threads, 57.7 GiB container RAM limit. The about-32 GB and about-16 GB conditions were created by other processes holding 25 and 41 GiB (leaving about 32.7 and 16.7 GiB). - Engine: the 0.15 rule, 25% cache, 15.7 GiB VRAM, measured with three read paths (SSD only, page cache first, automatic). - Baselines on the same machine: stock llama.cpp with the experts of 14 of 48 layers on the GPU (15.8 GiB VRAM), and with all experts on the CPU (2.8 GiB). - Measurement: prompts of 128, 512 and 1,536 tokens × 6 sessions, followed by the next 256 tokens of the data decoded one at a time (teacher forcing), session medians. Stock llama.cpp at 32 and 16 GB used 3 sessions. - Evidence - About 16 GB of RAM: the engine decoded 89.3–94.5 tokens/s, stock llama.cpp with the same VRAM 31.8, and with all experts on the CPU 16.0. Against the same-VRAM baseline this is 2.8–3.0× in decoding speed, and the same-VRAM baseline took 3.4–4.1× (page cache first) and 3.0–3.6× (SSD only) as long as the engine to answer (first token plus 256 tokens). - About 32 GB: ratios of 1.8 / 1.2 / 1.2 in time to answer (page cache first). The 1.8 for 128-token prompts may favor the engine: the stock configuration processed these prompts first and was slow on all three sessions, apparently while filling the page cache. - Ample RAM: 89.4–108.4 against 72.9 tokens/s in decoding (1.2–1.5×), and 1.4× in time to answer (page cache first). - Limits - One machine. On another machine with slower disk reads (4.3 GB/s, R12), the absolute values differed substantially. - With the SSD-only path, the engine took longer to answer than the same-VRAM baseline with ample RAM (6–17%) and at about 32 GB for 512- and 1,536-token prompts (7–12%); its decoding speed was higher in every condition. - The engine's perplexity was 1.6% higher than stock llama.cpp's (4,608 tokens). - The two policies of Claim 4 have not yet been compared with stock llama.cpp, and we did not search all stock configurations (layer placement, thread count). - At 16 GB, 25–32% of the experts were in the page cache, apparently filled when the model was loaded. On a real 16 GB PC other programs use RAM, so the gain may be smaller. - Section: §6.2

Claim 6. A planned generation test only partly supported Claims 2 and 4. [planned] - Conditions - Tasks: the first 500 of the 1,319 GSM8K test problems in a seed-14 shuffle, and all 164 HumanEval problems, in Qwen's chat template. - Generation: greedy, up to 1,024 tokens, each problem from an empty cache, prompts computed exactly. - Configurations: fetching every miss (reference), quality-first, speed-first, the 0.15 rule and the first-choice rule (all with 25% cache), and the first-choice rule with 12.5% cache. - Runs: three pods (two A40s, one RTX 3090) split the problems. Every paired comparison is within one pod. - Scoring: GSM8K uses the first number in the last $\square$ (else the last number in the response). A HumanEval answer is correct if its last closed Python code block (else everything after the last ```` ```python ````) passes the official tests within 10 s (the rule as implemented; read literally, the wording of the plan extracts different code in 2 responses and changes the outcome of 1, and no judgment changes). - Statistics: problem-level paired bootstrap (2,000 resamples) 95% intervals, exact McNemar tests and a non-inferiority margin of −2 points. Hypotheses and decision rules were fixed in a document before the runs. - Evidence - GSM8K accuracy differences (points) followed the order of mass: quality-first +0.4 [−1.4, +2.0], speed-first −0.2 [−2.0, +1.4], the 0.15 rule −1.8 [−3.6, 0.0], the first-choice rule −3.4 [−5.8, −1.2] (McNemar p = 0.0046), half cache −8.4 [−11.4, −5.6]. - Non-inferiority: quality-first on GSM8K only (lower bound −1.4; HumanEval −3.05), speed-first on both tasks (lower bounds −2.0 and −1.84). - Limits - Failed predictions: agreement following the order of mass on HumanEval (H2), the problem-level prediction (H3), accuracy differences of four configurations within −1.5 to +0.5 points (first part of H4, both tasks), and the absolute mass (1.8–2.4× the predicted values). Non-inferiority of quality-first on HumanEval (H5) was also not shown. - Speed-first's lower bounds sit at the margin and fall below it with another common interval method (Newcombe). With a different bootstrap seed, its HumanEval lower bound also fell below the margin (1 of 20 seeds). - Reference accuracy is near 94% on both tasks, so there is a ceiling effect, and each configuration ran one greedy pass. - We found a defect in the HumanEval extraction code during generation. Following an addendum to the plan written after the results were known, we regenerated all 984 outputs, confirmed that they matched the original records, and found that corrected extraction rules changed no outcome (§7.5). - Section: §7

Claim 7. A planned experiment that held mass fixed and varied only its composition reached no judgment under its planned rule. In the same run's auxiliary statistic and in exploratory conditions, a few heavy substitutions raised perplexity more than many light ones. [planned (no judgment) · auxiliary statistic and exploratory · emulation] - Conditions - Tool and data: the Qwen3-Next emulation of Claim 1, varying only the substitutions, without a cache; 48 sessions (8 per domain) chosen to avoid those of the earlier emulation and, presumably, engine measurements, 1,535 scored tokens each.

- Intervention: at each layer-token, among the nine experts in W other than the first choice, we pick two subsets whose masses are close to a target m* (0.10 in the primary condition) and to each other (within $0.01 \cdot m^*$). The heavy arm substitutes the subset whose weight is concentrated (1.44 experts on average), the light arm the one whose weight is spread out (2.16 experts). Layer-tokens without such a pair are not substituted. - Decision: the 95% session-bootstrap interval of R = Δppl_heavy / Δppl_light; no judgment if the interval for Δppl_light includes zero. - Evidence - Primary condition (planned): the light arm was +0.01% [−0.11, +0.13], so we made no judgment. The heavy arm was +0.43% [+0.11, +0.77], a difference of +0.41 [+0.13, +0.70] (a planned auxiliary statistic, not used for the judgment). Both arms realized a mean mass of 0.035. - Exploratory (run after seeing the results): at a target of 0.15, both arms realized a mass of 0.145; the heavy arm was +4.35% [+2.76, +6.09] and the light arm +0.49% [+0.19, +0.80]. With renormalization at a target of 0.10 and at a target of 0.05, the intervals of the difference also lay above zero (+0.39 and +0.07). - Limits - In the primary condition, a pair existed in only 33% of layer-tokens, so the realized mass was a third of the target; we had not computed this fraction from the model's logits before the run. - The heavy arm substitutes higher-ranked experts (mean rank 5.96 vs. 8.10), so weight and rank are not separated. - The planned decision rule did not handle a negative Δppl_light; at a target of 0.05 it output the opposite of what the data showed. - One model, BF16 emulation, teacher forcing. - Section: §5.4

What this paper does not claim - that gate weight itself causes the loss (we did not separate weight from rank, and Experiment B does not either); - that equal mass implies equal loss (in Experiment B, heavy substitutions lost more at equal mass); - that mass predicts which inputs will lose quality; - that emulation numbers carry over to the engine (the same 0.15 rule gave +0.35% and +1.21%); - that the engine results hold for other models (the engine was tested on Qwen3-Next only); - that our policies outperform prefetching or lower-precision methods (not compared); - that the speed ratios hold on other machines; - that free-form generation quality is preserved in general (the generation test covers greedy generation on two tasks).