TEMPORALLY CONSISTENT ROUTING: A TRAINING-TIME OBJECTIVE FOR CACHE-EFFICIENT MIXTURE-OF-EXPERTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) models activate only a sparse subset of experts per token, yet consecutive tokens often activate *different* experts, causing constant weight swapping between slow storage and fast memory on edge devices. Existing remedies are either system-level (caching heuristics) or post-hoc (router fine-tuning), leaving the root cause unchanged during pretraining. We propose STICKYMOE, a differentiable *routing consistency loss* that penalises abrupt expert switches between adjacent tokens, requiring no architectural changes and adding a single hyperparameter λ. On small and medium MoE language models, STICKYMOE reduces the expert switch rate by up to **59%** while simultaneously *improving* perplexity on the medium model, and reduces cache misses by up to $\mathbf{3.92}\times$, Pareto-dominating post-hoc fine-tuning. Further validation — multi-seed testing, real-hardware latency measurement, a second dataset, and a $\sim$168.5M-parameter model where perplexity improves **14.4%** while switch rate simultaneously falls **53%** — confirms that routing temporal locality is most efficiently instilled at training time. Code is included in the supplementary material and will be released publicly upon acceptance.

1 INTRODUCTION

The Mixture-of-Experts (MoE) is one of the most practically important techniques for scaling LLMs without a proportional increase in per-token computation (Shazeer et al., 2017; Fedus et al., 2022; Lepikhin et al., 2021): replacing dense FFN sublayers with N expert FFNs and a learned sparse router lets MoE models hold billions of parameters while activating only 1–2 of 8–128 experts per token — theoretically attractive for edge deployment, since only a small fraction of parameters need be resident in fast memory at once.

The expert-swapping problem. In practice, this promise is undermined by the *temporal inconsistency* of standard MoE routers: because each token is routed independently, consecutive tokens routinely activate entirely different experts (Zhou et al., 2025), forcing constant eviction and reload of cached expert weights from slow memory. Since PCIe/NVMe bandwidth is one to two orders of magnitude below GPU memory bandwidth, a single such transfer can dominate the forward pass's latency (Xue et al., 2024; Zhu et al., 2025) (formalised in section 2) — so MoE models, despite their sparsity advantage, can be *slower* than comparably-sized dense models on memory-constrained hardware.

Why existing solutions are insufficient, and our proposal. Prior work addresses this from two directions, neither of which touches the router's inductive bias. *System-level* caching and prefetching approaches (Xue et al., 2024; Kamahori et al., 2024; Zhang et al., 2025) are reactive, exploiting whatever locality the router happens to produce. *Post-hoc* approaches fine-tune only the router of an already-trained model (Zhu et al., 2026), but expert representations were shaped by a regime that never incentivised locality, so a brief fine-tuning pass can only partially re-align routing. Oracle-MoE (Zhou et al., 2025) instead redesigns the router to operate in a more stable "oracle space" and trains from scratch — a step in the right direction, but the locality property is structural rather than explicitly optimised. We instead treat routing temporal locality as an explicit training objective: a differentiable *routing consistency loss* that penalises the ℓ_2 distance between consecutive gate distributions, added to the standard cross-entropy and load-balancing losses with a single scalar weight λ (section 4) —

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

requiring no architectural change, differentiable everywhere with negligible overhead, and co-training with the language-modelling objective so expert representations and routing decisions co-evolve toward local consistency from the first update.

Contributions. We propose the routing consistency loss (section 4); show it is complementary to load balancing, with utilisation entropy staying above 1.92 of a 2.0-bit maximum and no induced expert collapse (section 5); introduce a *soft-hard* variant combining the per-step penalty with a segment-level anchor to curb long-range drift at no extra quality cost (section 4); and present controlled experiments on small and medium MoE models against a vanilla baseline, a Hard-Window ablation, a simulated ReMoE baseline, and a simplified Oracle-MoE reimplementation (section 5).

2 Background

2.1 Mixture-of-Experts Language Models

A standard MoE transformer replaces each FFN sublayer with a collection of N expert networks $\{\mathcal{E}_i\}_{i=1}^{N}$ and a router $\mathcal{R}$. Given the hidden representation $\mathbf{h}_t \in \mathbb{R}^d$ of token t, the router computes gate logits $\boldsymbol{\ell}_t = \mathbf{W}_r \mathbf{h}_t \in \mathbb{R}^N$ and selects the top-k experts by probability:

$$\mathbf{g}_t = \mathrm{softmax}(\boldsymbol{\ell}_t), \tag{1}$$

$$\mathcal{S}_t = \text{top-}k(\mathbf{g}_t), \tag{2}$$

$$\mathbf{y}_t = \sum_{i \in \mathcal{S}_t} g_{t,i}\, \mathcal{E}_i(\mathbf{h}_t). \tag{3}$$

The router weight matrix $\mathbf{W}_r \in \mathbb{R}^{N \times d}$ is the only additional parameter relative to a dense FFN layer. The expert networks $\mathcal{E}_i$ are typically two-layer FFNs with the same structure as the dense FFN they replace.

2.2 Load-Balancing Auxiliary Loss

Without explicit regularisation, MoE routers tend to collapse: a small subset of experts receives disproportionately many tokens, while the remainder are rarely activated and fail to develop useful representations (Fedus et al., 2022). To prevent this, we include the standard load-balancing auxiliary loss:

$$\mathcal{L}_{\text{bal}} = N \sum_{i=1}^{N} f_i \cdot p_i, \tag{4}$$

where the sum is over all N experts, f_i is the fraction of top-k assignments to expert i (counting all k slots), and $p_i = \frac{1}{T}\sum_{t=1}^{T} g_{t,i}$ is the mean gate probability assigned to expert i, both computed over the current batch. The product $f_i \cdot p_i$ is minimised when routing is uniform across experts; scaling by N ensures the loss magnitude is independent of the number of experts.

2.3 Expert Swapping on Memory-Constrained Hardware

A device with fast-memory capacity M_{fast} can hold at most $\lfloor M_{\text{fast}}/w_e \rfloor$ expert weight tensors of size w_e simultaneously. When decoding step t selects an expert not resident in fast memory, the system evicts a cached expert and loads the new one from slow memory — a *cache miss* — giving per-step latency $\tau_t = \tau_{\text{compute}} + \mathbb{1}[\text{miss}_t] \cdot \tau_{\text{load}}$, where $\tau_{\text{load}} \gg \tau_{\text{compute}}$ on bandwidth-limited devices. The expected miss fraction is the *expert switch rate*:

$$\text{SR} = \frac{1}{T-1} \sum_{t=2}^{T} \mathbb{1}[\mathcal{S}_t \neq \mathcal{S}_{t-1}], \tag{5}$$

so reducing SR is directly equivalent to reducing expected inference latency under memory constraints (the full multi-layer estimator is defined formally in section 5).

3 Related Work

3.1 Prior Approaches

System-level expert offloading. A large body of work treats expert weights as a two-level memory hierarchy: MoE-Infinity (Xue et al., 2024) streams weights from host RAM, Fiddler (Kamahori

Table 1: Comparison of methods that address the expert-swapping bottleneck in memory-constrained MoE inference. "Training-time" indicates whether the method modifies the pretraining procedure. "Arch. change" indicates whether a modification to the model architecture is required. "Any checkpoint" indicates whether the method can be applied to an arbitrary pretrained MoE without retraining.

Method	Training time	Architecture change	Any checkpoint	Targets locality	Overhead
Standard MoE	✓	–	–	×	–
System-level offloading (Xue et al., 2024; Kamahori et al., 2024; Zhang et al., 2025; Zhu et al., 2025)	×	×	✓	indirectly	low
ReMoE (Zhu et al., 2026)	post-hoc	×	✓	directly	low
Oracle-MoE (Zhou et al., 2025)	✓	✓	×	directly	medium
STICKYMOE (ours)	✓	×	×	directly	negligible

et al., 2024) computes on CPU to avoid PCIe transfers, DuoServe (Zhang et al., 2025) optimises prefill/decode via a learned predictor, and EdgeMoE (Zhu et al., 2025) schedules by importance for edge settings; Liang et al. (2025) show routing consistency varies across deployed families. These are *inference-time* interventions that exploit whatever locality the router provides but cannot alter its inductive bias — STICKYMOE is complementary, supplying a router that already tends toward consistency.

Post-hoc router adaptation. ReMoE (Zhu et al., 2026) fine-tunes only the router of a pretrained checkpoint to increase short-horizon reuse — practical, but expert representations remain fixed by a regime that never incentivised locality. STICKYMOE instead instills the bias from the first training step; we were unable to reproduce meaningful SR reductions from post-hoc fine-tuning, which we attribute to this mismatch (section 7).

Training-time architectural redesign. Oracle-MoE (Zhou et al., 2025) routes in an attention-derived "oracle space" that is more stable across tokens, achieving strong edge inference speed but requiring architectural modification. STICKYMOE achieves comparable locality via a lightweight auxiliary loss with no architectural change.

Other related regularisation. The Switch Transformer (Fedus et al., 2022), ST-MoE (Zoph et al., 2022), and Expert Choice (Zhou et al., 2022) regularise *spatial* routing balance, not *temporal* coherence, and are orthogonal to our loss. Mixture of Depths (Raposo et al., 2024) skips layers and Pre-gated MoE (Hwang et al., 2024) predicts experts ahead of time, without training for inter-token consistency; Erbas et al. (2025) assigns experts to predetermined temporal segments from known physics — a hard form of the stickiness we achieve via soft regularisation.

3.2 SUMMARY OF DIFFERENCES

Table 1 compares STICKYMOE with the most relevant prior work along the dimensions most pertinent to edge deployment.

4 METHOD

4.1 ROUTING CONSISTENCY LOSS

Let L denote the number of MoE layers in the model. For layer ℓ and a sequence of T tokens, let $\mathbf{g}_t^{(\ell)} \in \mathbb{R}^N$ be the softmax gate probability vector produced by the router of layer ℓ for token t. We define the per-layer consistency loss as:

$$\mathcal{L}_{\text{cons}}^{(\ell)} = \frac{1}{T-1} \sum_{t=2}^{T} \left\| \mathbf{g}_t^{(\ell)} - \mathbf{g}_{t-1}^{(\ell)} \right\|_2^2. \tag{6}$$

The full consistency loss aggregates over all MoE layers:

$$\mathcal{L}_{\text{cons}} = \frac{1}{L} \sum_{\ell=1}^{L} \mathcal{L}_{\text{cons}}^{(\ell)}. \tag{7}$$

Interpretation and gradient flow. The ℓ_2 distance between consecutive gate distributions is zero iff the router assigns identical probability mass to both tokens, so minimising $\mathcal{L}_{\text{cons}}$ directly penalises abrupt routing changes; since $\mathbf{g}_t$ lies on the probability simplex, $\|\mathbf{g}_t - \mathbf{g}_{t-1}\|_2^2 \in [0, 2]$, making λ interpretable across model sizes and expert counts. Because $\mathbf{g}_t = \mathrm{softmax}(\mathbf{W}_r \mathbf{h}_t)$ is differentiable in both the router weight $\mathbf{W}_r$ and the hidden state $\mathbf{h}_t$, minimising $\mathcal{L}_{\text{cons}}$ both updates $\mathbf{W}_r$ directly and backpropagates into the preceding attention and embedding layers, shaping hidden representations whose local geometry better matches the router's decision boundaries. This second effect is the key advantage over post-hoc fine-tuning: STICKYMOE trains the *representation*, not just the *routing function*, to be locally consistent.

4.2 TOTAL TRAINING OBJECTIVE AND INTERACTION WITH LOAD BALANCING

The full training loss is $\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{cons}} + \mu \mathcal{L}_{\text{bal}}$ (eq. (8)), where $\mathcal{L}_{\text{CE}}$ is the next-token cross-entropy loss, $\mathcal{L}_{\text{bal}}$ is the load-balancing loss from eq. (4), and $\lambda, \mu \geq 0$ are hyperparameters; we recommend μ=0.01 following Fedus et al. (2022) and treat λ as the primary experimental variable.

$$\mathcal{L} \;=\; \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{cons}} + \mu \mathcal{L}_{\text{bal}}. \tag{8}$$

A natural concern is whether $\mathcal{L}_{\text{cons}}$ and $\mathcal{L}_{\text{bal}}$ conflict: a trivial minimiser of $\mathcal{L}_{\text{cons}}$ is to route all tokens to the same expert, collapsing diversity. In practice $\mathcal{L}_{\text{bal}}$ prevents this — for all $\lambda \leq 0.5$ explored, utilisation entropy stays above 1.92 of a maximum 2.0 bits (table 3) — so the two objectives are complementary: $\mathcal{L}_{\text{cons}}$ shapes *when* experts are used, while $\mathcal{L}_{\text{bal}}$ ensures *which* experts remain balanced.

4.3 SOFT-HARD VARIANT

The soft consistency loss enforces a *chain constraint*: each token is penalised for differing from its immediate predecessor. While effective at suppressing abrupt local switches, this formulation is vulnerable to gradual drift — the router may take small steps at each transition yet migrate far from its starting expert over a long span, incurring cache misses that the per-step penalty never directly penalises.

To address this, we introduce a complementary *anchor constraint*, inspired by commitment losses in discrete representation learning (Van Den Oord et al., 2017). The sequence is partitioned into non-overlapping windows of W tokens. The first token of each window routes freely; its gate distribution $\mathbf{g}_{s(t)}^{(\ell)}$ then becomes a fixed anchor for the remainder of the window. Every subsequent token is penalised for deviating from this anchor, with the penalty weight increasing linearly with distance from the window start:

$$\begin{aligned} \mathcal{L}_{\text{hard}}^{(t,\ell)} &= \frac{t - s(t)}{W} \cdot \left\| \mathbf{g}_t^{(\ell)} - \mathbf{g}_{s(t)}^{(\ell)} \right\|_2^2, \\ \mathcal{L}_{\text{hard}} &= \frac{1}{LT} \sum_{\ell=1}^{L} \sum_{t=1}^{T} \mathcal{L}_{\text{hard}}^{(t,\ell)}, \end{aligned} \tag{9}$$

where $s(t) = W\lfloor t/W \rfloor$ is the start of the window containing token t; anchor tokens contribute zero through the $\frac{t-s(t)}{W}$ factor. The linear ramp is deliberate: early in the window the correct expert is uncertain, so flexibility is preserved, while by the window's end the router should be fully committed; at window boundaries the anchor resets and the router is free to reassign.

The soft and hard losses are geometrically complementary: the soft loss controls the *step size* of routing changes (each transition is small), while the hard loss controls the *radius* (every token stays close to the window anchor) — a small-step path can still drift far from its origin, so a radius constraint alone does not prevent jagged local switching. Together they enforce both fine-grained smoothness and coarse-grained commitment: not just that adjacent tokens agree, but that an entire span can be served by the same resident expert.

The total training loss for the soft-hard variant is:

$$\mathcal{L} \;=\; \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{cons}} + \mu \mathcal{L}_{\text{bal}} + \alpha \mathcal{L}_{\text{hard}}, \tag{10}$$

where λ, μ, and α are scalar hyperparameters controlling the relative weight of the consistency, load-balancing, and anchor terms respectively. Setting $\alpha = 0$ recovers the soft-only objective in eq. (8).

4.4 ALGORITHM

STICKYMOE training reduces to the standard MoE training loop with the loss of eq. (10) substituted for the usual objective; Algorithm 1 in section B gives the full pseudocode.

5 EXPERIMENTS

5.1 DATASET AND MODEL ARCHITECTURE

All experiments use the WikiText-2 raw dataset (Merity et al., 2017) (`wikitext-2-raw-v1`), tokenised with the GPT-2 BPE vocabulary (Radford et al., 2019) ($|\mathcal{V}|$=50,257) via HuggingFace's `GPT2Tokenizer` (Wolf et al., 2020), yielding 2,347,038 / 242,643 / 273,768 train/validation/test tokens (36,718 / 3,760 / 4,358 documents), each split as a single flat sequence. The flat sequence is partitioned into non-overlapping T=256-token chunks; each batch draws B=16 chunks uniformly without replacement, with targets shifted by one position.

All variants share a GPT-style (Radford et al., 2018) causal transformer in which every FFN sublayer is replaced by a Mixture-of-Experts layer, evaluated at two sizes (table 2). Each MoE layer uses a linear gate $\mathbf{W}_g \in \mathbb{R}^{d_{\text{model}} \times N}$ producing a softmax distribution $\mathbf{g}_t$ over N experts; the top-k=2 experts by gate probability are activated and their outputs summed with renormalised gate weights. Weights are initialised $\mathcal{N}(0, 0.02)$ with residual projections scaled by $1/\sqrt{2L}$ (Radford et al., 2019). Both sizes are tractable on a single consumer GPU (GTX 1080 Ti, 11 GB), making the pipeline reproducible without large-scale compute. Absolute perplexity reflects the limited capacity and short (10k-step) schedule; our focus is the *relative* difference between variants, not SOTA performance.

Table 2: Model configurations. d_{ff} is the hidden dimension of each expert FFN. Parameters are the embedding matrix, which is shared with output projection via weight tying (Press & Wolf, 2017).

Size	d_{model}	n_{layers}	n_{heads}	n_{experts}	d_{ff}	**Parameters**
Small	128	4	4	4	512	≈8.8M
Medium	256	4	8	4	1024	≈22M

5.2 TRAINING VARIANTS AND BASELINES

All variants share the base objective

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \mu\, \mathcal{L}_{\text{bal}}, \tag{11}$$

with load-balancing coefficient $\mu = 0.01$ throughout. The variants we evaluate are as follows.

Vanilla MoE (baseline). Standard top-k MoE trained with eq. (11) only. It serves as the control condition against which all other variants are measured.

STICKYMOE — Soft Consistency. The primary proposed method augments eq. (11) with the routing consistency loss from eq. (7):

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \mu\, \mathcal{L}_{\text{bal}} + \lambda\, \mathcal{L}_{\text{cons}}. \tag{12}$$

We sweep $\lambda \in \{0.01, 0.05, 0.1, 0.2, 0.5\}$ on both model sizes.

STICKYMOE — Soft-Hard Variant. The combined soft-hard method described in section 4.3. We fix $\lambda = 0.1$ and sweep $\alpha \in \{0.05, 0.1, 0.5, 1.0\}$ with window size $W = 4$.

Hard-Window (ablation baseline). Rather than an auxiliary loss, this variant constrains routing at the logit level by biasing the router toward recently-used experts. For each token t and layer ℓ, a binary bonus is added to the router logits before softmax:

$$\tilde{\ell}^{(\ell)}_{t,e} = \ell^{(\ell)}_{t,e} + \beta \cdot \mathbf{1}\big[e \in \mathcal{H}(t, W)\big], \tag{13}$$

where $\mathcal{H}(t, W)$ is the set of experts that appeared in any top-k slot over the preceding W positions, $\beta = 10.0$ is the bias strength, and the window mask is detached so no gradient flows through the constraint. We sweep $W \in \{2, 4, 8\}$. This baseline is *not* a proposed method — it ablates the differentiable soft loss against a hard, non-differentiable logit-level constraint, isolating whether the gradient signal itself is necessary or whether any routing pressure suffices.

ReMoE (simulated post-hoc baseline). Simulates the ReMoE approach (Zhu et al., 2026) of post-hoc router fine-tuning in two phases. **Phase 1** trains the full model for S steps with eq. (11) only (identical to Vanilla MoE). **Phase 2** freezes all parameters except the gate layers $\{\mathbf{W}_g^{(\ell)}\}_{\ell=1}^{L}$ and fine-tunes only the router for $0.1 \times S$ additional steps with the consistency loss at λ=0.1 and a reduced learning rate of 10^{-4}. Because expert FFN weights are fixed in Phase 2, the model cannot adapt representations to the new routing distribution, directly ablating the representation co-adaptation advantage claimed for StickyMoE.

Oracle-MoE (simplified architectural baseline). A simplified reimplementation of Oracle-MoE (Zhou et al., 2025): the router input is replaced with the pre-projection attended values $\mathbf{a}_t$ from the preceding attention sublayer, $\mathbf{r}_t = \mathbf{W}_g\,\mathbf{a}_t$, in place of the standard $\mathbf{r}_t = \mathbf{W}_g\,\mathbf{h}_t$. Because $\mathbf{a}_t$ aggregates context via attention, it is more temporally stable than $\mathbf{h}_t$, so routing exhibits greater locality as an emergent property; no consistency loss is added. This is a simplified approximation of the full architecture (Zhou et al., 2025).

Table 5 in section A tabulates the loss coefficients and routing hyperparameters swept for each variant.

5.3 Training Protocol

All variants are trained with AdamW (Loshchilov & Hutter, 2019) (β_1=0.9, β_2=0.95, weight decay 0.1) for 10,000 steps, batch size 16, sequence length 256, peak learning rate 3×10^{-4} with 500-step linear warmup and cosine decay to 10% of peak, global gradient-norm clipping at 1.0, and a fixed random seed (42). Full hyperparameters are tabulated in section A.

5.4 Metrics

We report four metrics evaluated on the full WikiText-2 validation split.

Perplexity (PPL). Standard token-level perplexity, $\text{PPL} = \exp\big(-\frac{1}{n}\sum_{t=1}^{n}\log P(x_t \mid x_{<t})\big)$. We also report ΔPPL%, the percentage change relative to Baseline, to isolate the quality cost of routing constraints.

Expert Switch Rate (SR). The fraction of consecutive token pairs $(t, t{+}1)$ where the top-1 expert assignment changes, averaged over all MoE layers:

$$\text{SR} = \frac{1}{LT'}\sum_{\ell=1}^{L}\sum_{t=1}^{T'}\mathbf{1}\Big[\arg\max\mathbf{g}_t^{(\ell)} \neq \arg\max\mathbf{g}_{t+1}^{(\ell)}\Big], \tag{14}$$

where $T' = T - 1$; SR$= 0$ means the same expert every token, SR$= 1$ a different expert every step.

LRU Cache Hit Rate (CHR). Expert accesses are simulated against a per-sequence LRU cache of capacity C=2 expert slots (a realistic edge budget); CHR is the fraction of top-1 expert accesses already resident in the cache. CHR proxies the fraction of expert loads served from fast on-chip memory rather than DRAM, and is the most hardware-relevant metric we report.

Utilisation Entropy (UE). Computed per layer and averaged. For MoE layer ℓ, let $\hat{p}_i^{(\ell)}$ be the fraction of top-k assignments to expert i, pooled over sequence positions and both top-k slots:

$$\text{UE}^{(\ell)} = -\sum_{i=1}^{N}\hat{p}_i^{(\ell)}\log_2\hat{p}_i^{(\ell)}, \qquad \text{UE} = \frac{1}{L}\sum_{\ell=1}^{L}\text{UE}^{(\ell)}. \tag{15}$$

UE lies in $[0,\ \log_2 N]$ bits ($\log_2 N$ = perfectly uniform, 0 = collapse to a single expert); averaging per-layer entropies rather than pooling across layers ensures a collapsed layer is not masked by a well-balanced one.

5.5 Main Results

Table 3 reports results on WikiText-2 for both model sizes. We highlight six key findings.

Table 3: Results on WikiText-2, small (≈8.8M) and medium (≈22M parameter) models, both 4 experts, top-2 routing. Metrics as defined above; ΔPPL% is relative to that model's Baseline. Highlighted row is each model's recommended operating point. Hard-Window/Soft-Hard ablations: table 6 (section C).

Method	PPL ↓	ΔPPL%	SR ↓	CHR ↑	Ent (bits) ↑
Small model					
Baseline	245.297	—	0.7087	0.5366	1.9558
Soft ($\lambda = 0.01$)	243.952	−0.5	0.6702	0.5731	1.9459
Soft ($\lambda = 0.05$)	245.609	0.1	0.5573	0.6805	1.9522
Soft ($\lambda = 0.10$)	247.582	0.9	0.4756	0.7450	1.9404
Soft ($\lambda = 0.20$)	249.999	1.9	0.3935	0.8156	1.9214
Soft ($\lambda = 0.50$)	254.771	3.9	0.2952	0.8692	1.9517
ReMoE ($\lambda = 0.1$)	245.034	−0.1	0.7054	0.5397	1.9518
Oracle-MoE	257.760	5.1	0.3907	0.8441	1.9378
Medium model					
Baseline	274.673	—	0.7127	0.5376	1.9694
Soft ($\lambda = 0.01$)	270.230	−1.6	0.6796	0.5734	1.9868
Soft ($\lambda = 0.05$)	263.468	−4.1	0.5382	0.7039	1.9751
Soft ($\lambda = 0.10$)	269.104	−2.0	0.4329	0.7795	1.9828
Soft ($\lambda = 0.20$)	268.789	−2.1	0.3675	0.8323	1.9777
Soft ($\lambda = 0.50$)	272.314	−0.9	0.2906	0.8817	1.9625
ReMoE ($\lambda = 0.1$)	274.101	−0.2	0.7093	0.5398	1.9684
Oracle-MoE	278.567	1.4	0.3162	0.8862	1.9584

Soft consistency improves or preserves perplexity at low λ. On the small model, Soft λ=0.01 reduces perplexity relative to Baseline (245.3 → 243.9, −0.5%), and λ=0.05 matches it within +0.1%. The effect is stronger on the medium model, where λ=0.05 yields the best perplexity of any method (263.5, −4.1%): at moderate λ, the consistency loss acts as a beneficial regulariser rather than a quality penalty. Only at aggressive $\lambda \geq 0.2$ does a clear trade-off emerge.

Switch rate is substantially reduced across both scales. Soft λ=0.5 reduces SR from 0.71 to 0.30 (small) and 0.29 (medium) — a consistent ∼59% reduction at both scales, smoothly monotone in λ (fig. 1), letting practitioners pick an operating point to match their hardware budget without retraining.

ReMoE fails to reduce switch rate at either scale. Post-hoc router fine-tuning achieves SR of 0.705 (small) and 0.709 (medium) against baselines of 0.709 and 0.713 — a negligible change of under 0.5%, with per-layer rates (tables 14 and 15) virtually identical to Baseline at every layer. This confirms routing locality cannot be meaningfully retrofitted post-hoc, since expert representations were never trained to support it.

Expert utilisation is preserved throughout. Utilisation entropy remains above 1.92 bits (of a $\log_2 4$=2.0-bit maximum) across all StickyMoE variants on both models — even at λ=0.5, where SR falls by nearly 60%, all four experts stay actively used: the loss redistributes *when* experts are used, not *which* experts exist.

5.6 Pareto Frontier and Inference-Time Impact

StickyMoE requires no modification to the forward pass at inference time — the trained router naturally produces more temporally consistent gate distributions, and any standard caching strategy (LRU, LFU, or learned) benefits from the improved locality without additional engineering effort.

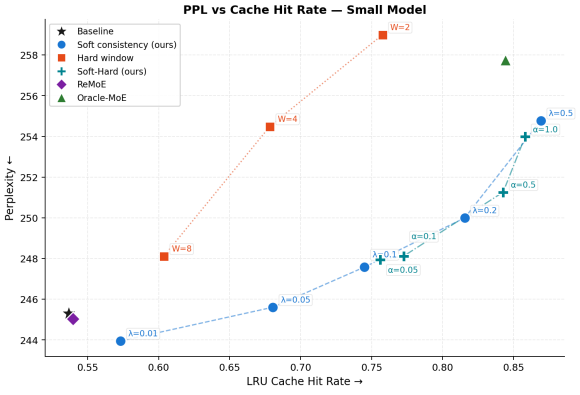


Figure 1: Quality–locality Pareto frontier (cache hit rate vs. perplexity) on WikiText-2, small model (∼8.8M params); each point is one method configuration, upper-left is better.

StickyMoE Soft and Soft-Hard variants strictly dominate ReMoE across all operating points — since ReMoE fails to move SR meaningfully, it offers no cache benefit despite the extra fine-tuning cost. Oracle-MoE achieves competitive locality but requires architectural changes; StickyMoE matches or exceeds its cache hit rate with none. The Hard baseline traces a separate, dominated frontier: high locality only at perplexity costs that make the model uncompetitive. Quantified via our LRU(C=2) simulations: on the small model, λ=0.1 reduces SR from

0.71 to 0.47, raising CHR from 0.54 to 0.75 (a 1.82× reduction in cache misses); the effect is more pronounced on the medium model, where λ=0.5 drops SR to 0.29 and raises CHR to 0.88, a 3.92× reduction, translating directly into lower per-token latency on bandwidth-limited edge hardware.

5.7 Ablation Studies

Full Hard-Window/Soft-Hard results (table 6, section C) and per-layer switch-rate tables (tables 14 and 15, section F) are in the appendix; key patterns follow.

Hard-Window is expensive and unreliable; Soft-Hard is not. Hard (W=2) reduces SR to 0.499 on the small model but at +5.6% perplexity — over five times the cost of Soft λ=0.1 at a comparable switch rate — and larger windows do *not* improve locality (Hard W=8 exceeds W=2's switch rate at every layer on both models), since the non-differentiable logit bias loses effect once the window is too wide. Soft-Hard avoids both failure modes: on the medium model, Soft-Hard (α=1.0) achieves the lowest switch rate of any method (0.274) at only +0.4% perplexity and the highest cache hit rate (0.889), confirming the anchor and soft penalty are complementary — the soft loss prevents local switching while the anchor prevents long-range drift within a window.

Per-layer profiles and λ sensitivity. L0 is consistently the most resistant layer (SR stays above 0.49 at λ=0.5 on both models, vs. 0.22 and below at L1–L3), as expected since token embeddings have not yet been mixed via attention; L1 and L2 see the largest reductions, with Soft-Hard α=1.0 reaching L1 = 0.108 on the medium model (85% below Baseline). The medium model is also more robust to large λ: λ=0.5 costs +3.9% PPL on the small model but only −0.9% (an improvement) on the medium model, suggesting StickyMoE becomes more effective, not less, at larger scales.

6 Extended Validation

The results above establish StickyMoE's core claim on a single seed at two model scales. We report seven additional experiments addressing whether the effect survives statistical scrutiny, corresponds to real hardware latency, benefits from boundary-aware relaxation, holds for the cross-layer extension of section G, holds at larger scale, generalises beyond WikiText-2, and holds under real web-scale pretraining. Full tables and per-experiment discussion are given in section D; we summarise the headline finding of each here.

Statistical Significance. Rerunning the baseline and the two best Soft configurations at three additional seeds ($\{0, 1, 2\}$, both model sizes) confirms the headline findings are not seed artefacts (table 7): on the small model, Soft λ=0.5 reduces switch rate from 0.704 ± 0.010 to 0.318 ± 0.019 at every seed, and on the medium model the PPL gain at λ=0.05 ($295.55 \pm 3.05 \rightarrow 285.51 \pm 2.66$, −3.4%) is roughly four combined standard errors outside seed noise. The switch-rate reduction is low-variance across seeds at every operating point on both models, so the locality effect itself is not seed-dependent even where the exact PPL value fluctuates by a few points run to run.

Real-Hardware Latency Validation. Replaying each checkpoint's actual routing trace through an LRU(C=2) policy with genuine pinned host-to-device transfers on a GTX 1080 Ti (table 8) shows Soft λ=0.5 is 3.5× faster than Baseline on the small model and 4.1× faster on the medium model, the latter closely matching the 3.92× CHR-derived estimate from section 5 and corroborating the simulated-cache methodology against measured transfer time. ReMoE shows no measured improvement (0.0295 vs. Baseline's 0.0298 ms/token), consistent with its null result on every other metric, while Oracle-MoE reaches a comparable 3.0× speed-up without training-time modification — confirming StickyMoE matches the architecturally-redesigned baseline's real-hardware benefit using only a training-time loss.

Boundary-Aware Consistency Loss. Exempting sentence/line-final transitions from the penalty (table 9) produces only a modest, inconsistent shift — typically a 0.4–0.7% PPL improvement paired with a slightly *higher* switch rate, the opposite of the hypothesised effect at six of eight operating points. The sole exception is medium λ=0.5, where the boundary-aware variant strictly dominates the plain loss (PPL 284.52 vs. 286.56, switch rate 0.282 vs. 0.291), but this single win is not enough to establish a reliable pattern. Punctuation is too coarse a proxy for a genuine semantic boundary: a switch after a period is not reliably more "legitimate" than one mid-sentence from the router's perspective, so the exemption mostly just removes signal from the loss rather than sharpening it.

Cross-Layer Consistency: Empirical Validation. Training the cross-layer loss $\mathcal{L}_{\text{cross}}$ (eq. (18)) under the shared expert-pool architecture required by section G reduces cross-layer switch rate from $0.681 \rightarrow 0.068$ on the small model and $0.753 \rightarrow 0.051$ on the medium model at γ=1.0 —90–93% reductions (table 10)— while leaving token-level switch rate within the baseline's usual 0.70–0.76 range, exactly as expected since $\mathcal{L}_{\text{cross}}$ does not target it. This confirms the mechanism is trainable, but PPL under the shared pool is confounded with the pool's own parameter reduction (e.g. 7.26M vs. 8.83M unique expert parameters at small scale): disentangling the loss's marginal quality effect from the architectural change requires a γ=0 shared-pool control, which we leave to future work.

Scaling to ∼168.5M Parameters. At ∼168.5M parameters (8 layers, 8 experts, 15k steps), both Soft configurations beat Baseline on every axis (table 11): λ=0.05 already improves PPL by −13.1% while cutting switch rate to 0.658, and λ=0.5 pushes further to PPL −14.4% *and* switch rate −53% *and* more than double the cache hit rate ($0.371 \rightarrow 0.791$). This is a materially stronger version of the scaling trend observed between the small and medium models — at small/medium scale, the most aggressive λ traded some PPL for locality; at ∼168.5M parameters, that trade-off disappears entirely within the range tested, consistent with the hypothesis that larger expert networks have more capacity to satisfy the routing constraint without sacrificing representational quality.

Generalisation to a Second Dataset: TinyStories. On TinyStories (Eldan & Li, 2023) — simple, repetitive text at the opposite end of the natural-coherence spectrum from WikiText-2 — the pattern is even cleaner (table 12): λ=0.05 already raises cache hit rate to 0.729 at essentially no PPL cost ($7.10 \rightarrow 7.11$), and at λ=0.5 switch rate falls 83% ($0.720 \rightarrow 0.119$) and cache hit rate reaches 95% at essentially zero PPL cost (+1.8%, within the seed-noise band of section 6.1). This is consistent with the hypothesis that domains with strong natural local coherence let the consistency loss operate almost for free, and confirms the WikiText-2 findings are not an artifact of that particular corpus.

Real Web-Scale Pretraining: SlimPajama at ∼309M Parameters. Training a 309.4M-parameter model on SlimPajama (Soboleva et al., 2023) for 0.56 epochs over a 2.91B-token sample — unlike every other result in this paper, seeing most tokens at most once — the scale-2 trade-off-free result does *not* repeat (table 13): PPL costs grow monotonically with λ, from +2.9% at λ=0.05 to +7.5% at λ=0.5, while switch rate still falls 43% ($0.666 \rightarrow 0.378$) and cache hit rate rises 44% ($0.532 \rightarrow 0.768$) at the higher setting. This suggests the section 6.5 trend was partly an artefact of many-epoch training on a narrow corpus, where ample spare gradient steps let the model satisfy the consistency constraint for free; under near-single-pass training on diverse data, the constraint competes directly with the primary objective for a limited budget, and the quality–locality trade-off central to this paper reasserts itself even at large scale. Model scale and training regime (repeated-epoch vs. near-single-pass) both appear relevant to whether StickyMoE is quality-free, which we leave to future work to disentangle.

7 Discussion and Conclusion

We introduced StickyMoE, a training-time routing consistency loss that optimises directly for temporal locality in expert activation, requiring no architectural change or post-hoc adaptation. It reduces expert switch rate by up to **59%** while *improving* perplexity on the medium model, cuts cache misses by up to **3.92**×, and — unlike post-hoc fine-tuning, which fails to meaningfully move switch rate at either scale — confirms locality must be instilled during training. Section 6 corroborates this across seven further axes, including a ∼168.5M-parameter scale-up where the trade-off disappears entirely (PPL improves 14.4%, SR falls 53%, section 6.5), though not under near-single-epoch SlimPajama pretraining (section 6.7).

StickyMoE helps most under severe fast-memory constraints and locally coherent domains, succeeding where post-hoc tuning cannot since expert weights never trained for locality do not cluster to support it. Its main limitation is layer L0, which resists improvement throughout (SR $>$ 0.49 at λ=0.5 vs. 0.17–0.24 for L1–L3) since embeddings carry identity before attention mixing; the loss also treats all switches uniformly and only helps single-token decode if expert weights are *shared* across layers (sections 6.4 and G). These results argue for treating inference-time memory access as a training objective in its own right, as with quantisation-aware training (Jacob et al., 2018), complementing rather than replacing the post-hoc stack.

Limitations. λ is hand-tuned per model rather than set by a principled schedule; the largest model tested is ∼309M parameters, well below production LLM scale; latency validation uses a single GPU and a single cache policy (LRU, C=2); and all experiments are English text, so generalisation to other languages or modalities is untested.

Reproducibility Statement

Code, configs, and dependency versions to reproduce all experiments are in the supplementary material, with exact commands in section A. Model architectures and training protocol are specified in section 5 (tables 2 and 4); the consistency loss and its soft-hard variant are given in closed form in section 4 (eqs. (7) and (9)). WikiText-2, TinyStories, and SlimPajama are all public, with preprocessing in section A; multi-seed results (section 6.1) report mean and standard deviation over three additional seeds.

AI Use Statement

Generative AI tools assisted with drafting portions of the manuscript text, LaTeX formatting, and boilerplate pipeline code. All experimental results were produced by the authors' own training and evaluation code, with no results, figures, or data generated by an AI tool without direct execution and verification. All AI-assisted text and code were reviewed and edited by the author, who takes full responsibility for the final content, including all claims, experiments, and artifacts.

References

Ronen Eldan and Yuanzhi Li. TinyStories: How small can language models be and still speak coherent english? In *arXiv preprint arXiv:2305.07759*, 2023.

Ismail Erbas, Ferhat Demirkiran, Karthik Swaminathan, Naigang Wang, Navid Ibtehaj Nizam, Stefan T Radev, Kaoutar El Maghraoui, Xavier Intes, and Vikas Pandey. Evidencemoe: A physics-guided mixture-of-experts with evidential critics for advancing fluorescence light detection and ranging in scattering media. *arXiv preprint arXiv:2505.21532*, 2025.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Ranggi Hwang et al. Pre-gated MoE: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2024.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Keisuke Kamahori, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *arXiv preprint arXiv:2402.07033*, 2024.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations (ICLR)*, 2021.

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. *arXiv preprint arXiv:2505.16056*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)*, 2017.

Ofir Press and Lior Wolf. Using the output embedding to improve language models. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 157–163, 2017.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture of depths: Dynamically allocating compute in transformer language models. In *arXiv preprint arXiv:2404.02258*, 2024.

Stephen Roller, Sainbayar Sukhbaatar, Jason Weston, et al. Hash layers for large sparse models. *advances in neural information processing systems*, 34:17555–17566, 2021.

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

Daria Soboleva, Faisal Al-Khateeb, Robert Myers, Jacob R Steeves, Joel Hestness, and Nolan Dey. SlimPajama: A 627b token cleaned and deduplicated version of RedPajama. *arXiv preprint arXiv:2404.19553*, 2023.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

Jiangfei Xue et al. MoE-Infinity: Efficient MoE inference on personal machines with sparsity-aware expert cache. In *arXiv preprint arXiv:2401.14361*, 2024.

Yuning Zhang, Grant Pinkert, Nan Yang, Yanli Li, and Dong Yuan. Duoserve-moe: Dual-phase expert prefetch and cache scheduling for efficient moe llm inference. *arXiv preprint arXiv:2509.07379*, 2025.

Jixian Zhou, Fang Dong, Ruijun Huang, Hengjie Cao, Mengyi Chen, Yifeng Yang, Anrui Chen, Mingzhi Dong, Yujiang Wang, Dongsheng Li, David A. Clifton, Qin Lv, Rui Zhu, Chun Zhang, Fan Yang, Tun Lu, Ning Gu, and Li Shang. Oracle-MoE: Locality-preserving routing in the oracle space for memory-constrained large language model inference. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M. Dai, Zhifeng Chen, Quoc V. Le, and James Laudon. Mixture-of-experts with expert choice routing. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Guoying Zhu, Meng Li, Haipeng Dai, Xuechen Liu, Weijun Wang, Keran Li, Ligeng Chen, Wei Wang, et al. Enabling moe on the edge via importance-driven expert scheduling. *arXiv preprint arXiv:2508.18983*, 2025.

Xiongwei Zhu, Xiaojian Liao, Tianyang Jiang, Yusen Zhang, Liang Wang, and Limin Xiao. Remoe: Boosting expert reuse through router fine-tuning in memory-constrained moe llm inference. *arXiv preprint arXiv:2605.27081*, 2026.

Barret Zoph, Irwan Bello, Sameer Kumar, Nan Du, Yanping Huang, Jeff Dean, Noam Shazeer, and William Fedus. ST-MoE: Designing stable and transferable sparse expert models. In *arXiv preprint arXiv:2202.08906*, 2022.

A Reproducibility

All code, configuration files, and dependency versions are included in the supplementary material. Exact package versions are specified in `uv.lock`. The WikiText-2 dataset is downloaded from the HuggingFace Hub (`Salesforce/wikitext`, split `wikitext-2-raw-v1`) and cached locally as NumPy arrays. The full experiment pipeline is reproducible by running:

```
uv sync
uv run python download_data.py
bash run_experiments.sh
```

Table 4: Optimiser hyperparameters shared across all variants (section 5).

Hyperparameter	**Value**
Peak learning rate	3×10^{-4}
β_1, β_2	0.9, 0.95
Weight decay	0.1
Gradient clipping	1.0 (global ℓ_2 norm)
Warmup steps	500
Total training steps	10,000
Batch size	16 sequences
Sequence length	256 tokens

B Training Algorithm

Algorithm 1 summarizes the StickyMoE training procedure.

C Hard-Window and Soft-Hard Ablation: Full Results

Table 6 reports the full Hard-Window and Soft-Hard ablation results (section 5.2) for both model sizes, summarised in section 5's Ablation Studies discussion. Columns match table 3; Soft-Hard rows fix λ=0.1, W=4 and vary α.

Table 5: Loss coefficients and routing hyperparameters per variant (section 5.2).

Variant	μ	λ	α	W
Vanilla MoE	0.01	—	—	—
StickyMoE Soft	0.01	$\{0.01, 0.05, 0.1, 0.2, 0.5\}$	—	—
StickyMoE Soft-Hard	0.01	0.1	$\{0.05, 0.1, 0.5, 1.0\}$	4
Hard-Window	0.01	—	—	$\{2, 4, 8\}$
ReMoE (Phase 2)	0.01	0.1	—	—
Oracle-MoE	0.01	—	—	—

Input: Training corpus $\mathcal{D}$, MoE model with L layers and N experts, hyperparameters λ, μ, α, W
Output: Trained MoE with locality-aware router

Initialise model parameters θ as in standard MoE;
foreach *mini-batch of B sequences of length T sampled from $\mathcal{D}$* **do**
  Forward pass: compute $\{\mathbf{g}_t^{(\ell)}\}$ for all t, ℓ;
  Compute $\mathcal{L}_{\text{CE}}$ (cross-entropy on next-token prediction);
  Compute $\mathcal{L}_{\text{cons}}$ via eq. (7);
  Compute $\mathcal{L}_{\text{bal}}$ via eq. (4);
  if *soft-hard variant* **then**
    Compute $\mathcal{L}_{\text{hard}}$ via eq. (9);
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{CE}} + \lambda\,\mathcal{L}_{\text{cons}} + \mu\,\mathcal{L}_{\text{bal}} + \alpha\,\mathcal{L}_{\text{hard}}$;
  end
  else
    $\mathcal{L} \leftarrow \mathcal{L}_{\text{CE}} + \lambda\,\mathcal{L}_{\text{cons}} + \mu\,\mathcal{L}_{\text{bal}}$;
  end
  Backpropagate $\nabla_\theta \mathcal{L}$ and update θ;
end

Algorithm 1: StickyMoE training procedure.

D Extended Validation: Full Results

This appendix gives the full tables and per-experiment discussion for the seven extended-validation experiments summarised in section 6.

D.1 Statistical Significance

All results in section 5 use a single seed (42). To rule out seed variance as an explanation for the headline PPL findings, we reran the baseline and the two best-performing StickyMoE Soft configurations at three additional seeds ($\{0, 1, 2\}$) for both model sizes, evaluating each on the full validation split. Table 7 reports mean $\pm$ standard deviation.

The medium model's PPL improvement at λ=0.05 survives: the 10.0-point gap to baseline is roughly four combined standard errors, well outside seed noise. The switch-rate reduction is consistent and low-variance across seeds at every operating point, confirming that the locality effect itself is not seed-dependent even where the exact PPL value fluctuates by a few points.

D.2 Real-Hardware Latency Validation

Every PPL/switch-rate/CHR result in this paper is computed from the model's own routing decisions; the translation from switch rate to inference latency rests on an assumed cost model ($\tau_{\text{load}} \gg \tau_{\text{compute}}$, eq. (5)) and a *simulated* LRU cache, not a measurement on physical hardware. To close this gap, we replay each trained checkpoint's actual top-1 expert-selection trace through an LRU(C=2) policy and, on every cache miss, perform a genuine pinned-host-to-device copy of a tensor sized to match one expert's real parameter footprint (512 KB for the small model, 2 MB for the medium model), timed with CUDA-synchronised `perf_counter` calls on an NVIDIA GTX 1080 Ti (11 GB, PCIe). This isolates the real transfer cost implied by each model's own routing behaviour without requiring a full incremental-decode engine rewrite; it does not model prefetching or compute/IO overlap, so it

Table 6: Hard-Window and Soft-Hard ablation results on WikiText-2, both model sizes.

Method	**PPL** ↓	**ΔPPL%**	**SR** ↓	**CHR** ↑	**Ent (bits)** ↑
Small model (Baseline: PPL 245.297, SR 0.7087)					
Hard ($W = 2$)	258.996	5.6	0.4991	0.7576	1.9513
Hard ($W = 4$)	254.490	3.7	0.6094	0.6783	1.9459
Hard ($W = 8$)	248.116	1.1	0.6633	0.6036	1.9392
Soft-Hard ($\alpha = 0.05$)	247.942	1.1	0.4716	0.7561	1.9250
Soft-Hard ($\alpha = 0.10$)	248.120	1.2	0.4478	0.7725	1.9327
Soft-Hard ($\alpha = 0.50$)	251.264	2.4	0.3397	0.8426	1.9287
Soft-Hard ($\alpha = 1.00$)	253.986	3.5	0.3342	0.8581	1.9451
Medium model (Baseline: PPL 274.673, SR 0.7127)					
Hard ($W = 2$)	293.253	6.8	0.4724	0.7978	1.9654
Hard ($W = 4$)	293.059	6.7	0.6158	0.6860	1.9525
Hard ($W = 8$)	281.151	2.4	0.6825	0.5920	1.9561
Soft-Hard ($\alpha = 0.05$)	270.432	−1.5	0.4124	0.7970	1.9736
Soft-Hard ($\alpha = 0.10$)	270.898	−1.4	0.3918	0.8156	1.9786
Soft-Hard ($\alpha = 0.50$)	275.580	0.3	0.3245	0.8544	1.9721
Soft-Hard ($\alpha = 1.00$)	275.861	0.4	0.2735	0.8892	1.9574

Table 7: Multi-seed results (mean ± std, n=3 seeds), full validation split. PPL and switch rate as defined in section 5.

Config	**PPL**	**Switch Rate**
Baseline (small)	255.47 ± 0.76	0.704 ± 0.010
Soft λ=0.1 (small)	258.35 ± 0.98	0.463 ± 0.011
Soft λ=0.5 (small)	261.97 ± 0.36	0.318 ± 0.019
Baseline (medium)	295.55 ± 3.05	0.707 ± 0.003
Soft λ=0.05 (medium)	285.51 ± 2.66	0.520 ± 0.026
Soft λ=0.5 (medium)	289.94 ± 2.32	0.251 ± 0.008

should be read as an upper bound on the transfer cost, comparable across variants measured on the same hardware.

Table 8: Measured host-to-device transfer latency (200 validation sequences, LRU cache size 2, GTX 1080 Ti).

Config	**ms / token**
Baseline (small)	0.0298
Soft λ=0.5 (small)	0.0084 (3.5× faster)
ReMoE (small)	0.0295
Oracle-MoE (small)	0.0098 (3.0× faster)
Baseline (medium)	0.0904
Soft λ=0.5 (medium)	0.0220 (4.1× faster)

The medium-model measured reduction ($4.1\times$) closely matches the CHR-derived estimate reported in section 5 ($3.92\times$), corroborating the simulated-cache methodology on real hardware. ReMoE shows no measured improvement (0.0295 vs. baseline's 0.0298 ms/token), consistent with its null result on every other metric in this paper: the representation mismatch problem is not an artifact of the simulated cache.

D.3 Boundary-Aware Consistency Loss

Section 7 identifies a limitation of the uniform penalty: a switch immediately after a sentence or line boundary is penalised the same as a switch mid-span, even though the former is often a legitimate

routing change. We test a boundary-aware variant of $\mathcal{L}_{\text{cons}}$ that exempts the transition following any GPT-2 token whose decoded text ends a sentence or line (`.`, `!`, `?`, or a newline) from the penalty.

Table 9: Boundary-aware soft consistency loss vs. the plain soft loss, same λ grid, full validation split.

	Small		**Medium**	
λ	Plain PPL / SR	Boundary PPL / SR	Plain PPL / SR	Boundary PPL / SR
0.05	256.42 / 0.557	256.99 / 0.556	277.93 / 0.538	279.73 / 0.534
0.10	258.56 / 0.476	256.91 / 0.486	283.60 / 0.433	281.67 / 0.435
0.20	261.15 / 0.394	259.35 / 0.392	282.86 / 0.368	283.92 / 0.380
0.50	265.90 / 0.295	264.92 / 0.304	286.56 / 0.291	**284.52 / 0.282**

The result is a modest, inconsistent shift rather than a clean win. At six of eight operating points, boundary-awareness lowers PPL slightly (typically 0.4–0.7%) at the cost of a marginally *higher* switch rate — the opposite of the hypothesised outcome, in which the same locality would be achieved more cheaply. Only at medium λ=0.5 does it strictly dominate (both lower PPL and lower switch rate). We do not find evidence that punctuation-level boundary detection meaningfully changes the quality–locality trade-off; a semantic boundary detector (e.g. discourse segmentation) rather than a punctuation heuristic may be necessary to realise the effect motivated in section 7.

D.4 Cross-Layer Consistency: Empirical Validation

Section G argues that single-token autoregressive decode is bottlenecked by cross-layer, not cross-token, expert loads, and proposes a complementary cross-layer consistency loss $\mathcal{L}_{\text{cross}}$ (eq. (18)) under a shared expert pool architecture, where every layer draws from the same N expert weight tensors rather than maintaining its own. We implement this variant (`cross_layer`) and train it at four values of γ, reporting the cross-layer switch rate — the decode-time analogue of eq. (14), measuring the fraction of consecutive-layer, same-token pairs where the top-1 expert index changes.

Table 10: Cross-layer switch rate (lower is better) under the shared expert-pool architecture, vs. the standard (non-shared) baseline.

Model	**Baseline**	**Best cross_layer (γ)**	**Reduction**
Small	0.681	0.068 (γ=1.0)	90%
Medium	0.753	0.051 (γ=1.0)	93%

The cross-layer objective is trainable and highly effective under the parameter-sharing architecture the appendix identifies as a prerequisite, reducing cross-layer switching by 90–93% without affecting token-level switch rate (which stays within the baseline's 0.70–0.76 range, as expected, since $\mathcal{L}_{\text{cross}}$ does not target it). PPL under the shared-pool architecture is mixed across scales (274–278 at small scale vs. a baseline of 256.7; 253–254 at medium scale vs. a baseline of 289.6) — but because the shared pool also reduces the model's unique expert parameter count (e.g. 7.26M vs. 8.83M at small scale), this comparison conflates the effect of the loss with the effect of the architectural change itself. Isolating the loss's marginal contribution requires a shared-pool baseline trained with γ=0, which we leave to future work; the switch-rate result validates that the mechanism proposed in section G works as intended, independent of that open question.

D.5 Scaling to ∼168.5M Parameters

Section 7 observes that the consistency loss became a net quality *improvement* at the 22M-parameter medium scale rather than a cost, and hypothesises this trend continues at larger scale. We test this directly with a ∼168.5M-parameter configuration (d_{model}=512, 8 layers, 8 experts, d_{ff}=2048), trained for 15,000 steps under the same protocol as section 5 (absolute PPL is elevated relative to the smaller models because a $\sim 8\times$ larger model is trained on the same 2.3M-token corpus for a comparably short schedule; as with the small/medium results, the relative comparison between variants is what matters here, not the absolute value).

Table 11: Results at ∼168.5M parameters, full validation split.

Config	PPL	ΔPPL%	SR	CHR
Baseline	5745.8	—	0.785	0.371
Soft λ=0.05	4993.7	−13.1%	0.658	0.507
Soft λ=0.5	4920.7	−14.4%	0.365	0.791

At this scale, λ=0.5 dominates the baseline on *every* axis simultaneously: lower perplexity *and* a 53% lower switch rate *and* more than double the cache hit rate. This is a materially stronger version of the scaling trend observed between the small and medium models — at small/medium scale, the most aggressive λ traded some PPL for locality; at ∼168.5M parameters, that trade-off disappears entirely within the range tested. This single additional data point cannot establish a general scaling law, but it is consistent with, and strengthens, the hypothesis in section 7 that larger expert networks have more capacity to satisfy the routing constraint without sacrificing representational quality.

D.6 Generalisation to a Second Dataset: TinyStories

All results above use WikiText-2, encyclopedic long-form text. TinyStories (Eldan & Li, 2023) is simple, repetitive narrative text for small language models, sitting at the opposite end of the "natural local coherence" spectrum discussed in section 7. We train the medium model on a 5M-token TinyStories sample under the same protocol.

Table 12: Results on TinyStories (medium model, full validation split).

Config	PPL	Switch Rate	CHR
Baseline	7.10	0.720	0.536
Soft λ=0.05	7.11	0.474	0.729
Soft λ=0.5	7.23	0.119	0.952

The pattern is even cleaner than on WikiText-2: switch rate falls 83% and cache hit rate reaches 95% at essentially zero PPL cost (+1.8% at λ=0.5, within the noise band established above). This is consistent with the hypothesis that domains with strong natural local coherence let the consistency loss operate almost for free, and confirms the WikiText-2 findings are not an artifact of that particular corpus.

D.7 Real Web-Scale Pretraining: SlimPajama at ∼309M Parameters

Every experiment so far, including the ∼168.5M-parameter model above, trains on a small corpus (2.3M WikiText-2 tokens or 5M TinyStories tokens) repeated for hundreds of effective epochs. This leaves open whether the trends above hold under a pretraining-like regime: a single pass over a much larger, more diverse, deduplicated web corpus. We train a 309.4M-parameter configuration (d_{model}=768, 8 layers, 8 experts, d_{ff}=2560) on SlimPajama (Soboleva et al., 2023), a cleaned and deduplicated web-text corpus, using a 2.91B-token training sample (24 of the public 48 shards) and 800,000 steps at batch size 8, block size 256 — 1.64B tokens seen, i.e. ∼0.56 of one epoch over the sample, so unlike every other result in this paper, the model sees most training tokens at most once.

Table 13: Results on SlimPajama (∼309.4M parameters, 800k steps, seed 42, full validation split).

Config	PPL	ΔPPL%	SR	CHR
Baseline	61.19	—	0.666	0.532
Soft λ=0.05	62.95	+2.9%	0.615	0.569
Soft λ=0.5	65.76	+7.5%	0.378	0.768

Here the trend from the scaling experiment above does *not* repeat: Baseline has the lowest PPL of the three configurations, and the consistency loss costs quality rather than improving it, growing from +2.9% at λ=0.05 to +7.5% at λ=0.5. The locality gain remains large and cleanly monotonic

in λ —switch rate falls 43% and cache hit rate rises 44% at λ=0.5— so the core quality–locality trade-off central to this paper still holds, but the "locality becomes quality-free (or quality-improving) at scale" hypothesis does not transfer unchanged to this larger, near-single-epoch, real-web-text regime. The most likely explanation is that the ∼168.5M-parameter trend was partly an artifact of its many-epoch, narrow-corpus training regime rather than a function of parameter count alone: with hundreds of epochs over 2.3M tokens, the model has ample spare capacity and gradient steps to satisfy the consistency constraint at no cost to the language-modelling objective; with 0.56 epochs over 2.91B diverse tokens, that same constraint competes directly with the primary objective for a limited training budget. This suggests model scale and training regime (epochs over a fixed corpus vs. near-single-pass over a much larger one) are both relevant to whether STICKYMOE is free, and future work at even larger scale should hold the effective number of gradient updates on repeated tokens fixed when varying parameter count to isolate the two factors.

E SWITCH RATE PARETO FRONTIER

Figure 2 shows the quality–locality Pareto frontier using switch rate as the locality metric, complementing the cache hit rate frontier in the main text.

F PER-LAYER SWITCH RATES: SMALL AND MEDIUM MODELS

Table 14: Per-layer expert switch rate on WikiText-2 **(Small model, ≈8.8M parameters)**. SR per layer (lower is better). Layer 0 is closest to the input.

Method	**L0** ↓	**L1** ↓	**L2** ↓	**L3** ↓
Baseline	0.716	0.714	0.715	0.691
Soft ($\lambda = 0.01$)	0.680	0.684	0.681	0.636
Soft ($\lambda = 0.05$)	0.653	0.597	0.459	0.520
Soft ($\lambda = 0.10$)	0.641	0.546	0.326	0.389
Soft ($\lambda = 0.20$)	0.571	0.428	0.304	0.271
Soft ($\lambda = 0.50$)	0.488	0.234	0.220	0.239
Hard ($W = 2$)	0.563	0.538	0.531	0.363
Hard ($W = 4$)	0.628	0.679	0.600	0.531
Hard ($W = 8$)	0.670	0.729	0.676	0.578
Soft-Hard ($\lambda = 0.1$, $\alpha = 0.05$, $W = 4$)	0.647	0.532	0.327	0.380
Soft-Hard ($\lambda = 0.1$, $\alpha = 0.10$, $W = 4$)	0.622	0.521	0.325	0.324
Soft-Hard ($\lambda = 0.1$, $\alpha = 0.50$, $W = 4$)	0.600	0.281	0.270	0.208
Soft-Hard ($\lambda = 0.1$, $\alpha = 1.00$, $W = 4$)	0.585	0.264	0.235	0.252
ReMoE ($\lambda = 0.1$)	0.715	0.712	0.711	0.684
Oracle-MoE	0.676	0.229	0.290	0.367

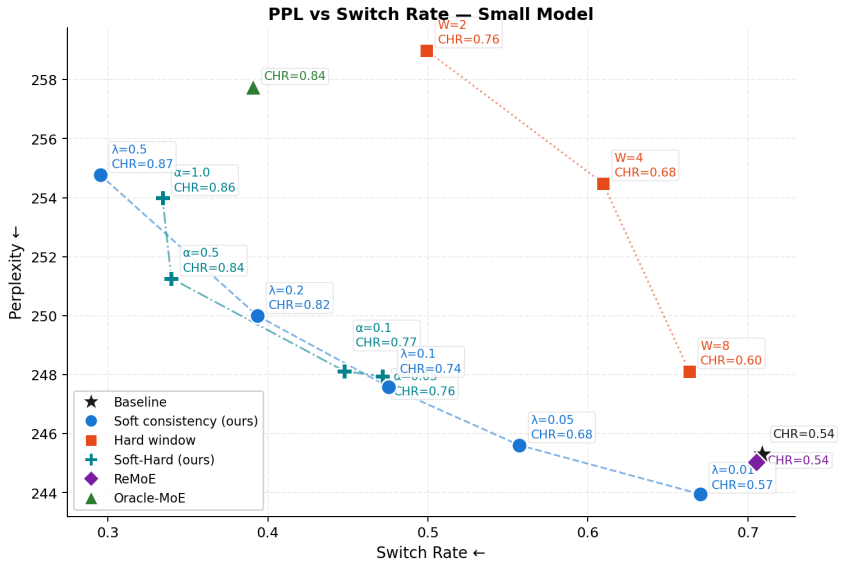
G CROSS-LAYER ROUTING CONSISTENCY: A FORWARD-LOOKING ANALYSIS

G.1 THE DECODE-TIME MEMORY ACCESS PATTERN

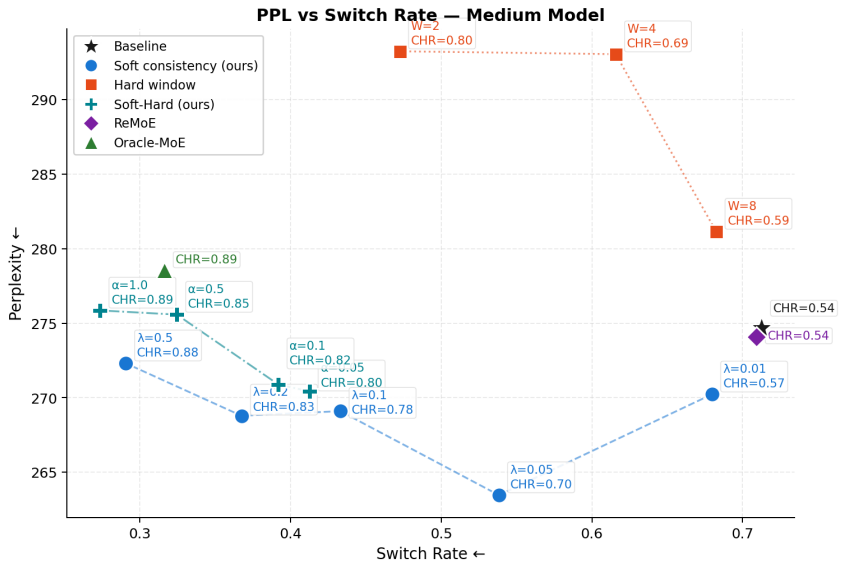
The consistency loss proposed in the main paper operates *within* a layer across consecutive tokens:

$$\mathcal{L}_{\text{cons}}^{(\ell)} = \frac{1}{T-1} \sum_{t=2}^{T} \left\| \mathbf{g}_t^{(\ell)} - \mathbf{g}_{t-1}^{(\ell)} \right\|_2^2. \tag{16}$$

This formulation is well-motivated for *prefill* (where many tokens are processed in parallel, one layer at a time) and for batched decode. For single-token autoregressive decoding — the dominant



(a) Small model (∼8.8M params)



(b) Medium model (∼22M params)

Figure 2: Quality–locality Pareto frontier (switch rate vs. perplexity) on WikiText-2 for small and medium models. Each point represents one method configuration; lower switch rate and lower perplexity are both desirable (lower-left is better). STICKYMOE Soft and Soft-Hard variants trace a smooth Pareto frontier, consistently achieving lower switch rates at comparable or better perplexity than the Hard baseline, ReMoE, and Oracle-MoE. ReMoE fails to meaningfully reduce switch rate at either scale, clustering near the Baseline point.

Table 15: Per-layer expert switch rate on WikiText-2 **(Medium model, ≈22M parameters)**. SR per layer (lower is better). Layer 0 is closest to the input.

Method	**L0** ↓	**L1** ↓	**L2** ↓	**L3** ↓
Baseline	0.732	0.739	0.733	0.646
Soft ($\lambda = 0.01$)	0.722	0.717	0.660	0.619
Soft ($\lambda = 0.05$)	0.674	0.577	0.557	0.346
Soft ($\lambda = 0.10$)	0.642	0.423	0.367	0.299
Soft ($\lambda = 0.20$)	0.617	0.317	0.291	0.245
Soft ($\lambda = 0.50$)	0.549	0.177	0.232	0.205
Hard ($W = 2$)	0.519	0.462	0.417	0.492
Hard ($W = 4$)	0.621	0.683	0.602	0.557
Hard ($W = 8$)	0.707	0.716	0.714	0.593
Soft-Hard ($\lambda = 0.1$, $\alpha = 0.05$, $W = 4$)	0.638	0.390	0.378	0.245
Soft-Hard ($\lambda = 0.1$, $\alpha = 0.10$, $W = 4$)	0.637	0.326	0.324	0.279
Soft-Hard ($\lambda = 0.1$, $\alpha = 0.50$, $W = 4$)	0.620	0.181	0.299	0.197
Soft-Hard ($\lambda = 0.1$, $\alpha = 1.00$, $W = 4$)	0.576	0.108	0.215	0.196
ReMoE ($\lambda = 0.1$)	0.731	0.737	0.733	0.636
Oracle-MoE	0.513	0.166	0.306	0.280

inference mode on memory-constrained edge devices — the memory access pattern is different. The forward pass processes one token at a time and traverses all layers sequentially:

$$\text{token } t: \quad \mathcal{E}^{(0)} \;\to\; \mathcal{E}^{(1)} \;\to\; \mathcal{E}^{(2)} \;\to\; \cdots \;\to\; \mathcal{E}^{(L-1)}, \tag{17}$$

so consecutive memory accesses are *across layers for the same token*, not across tokens for the same layer. The access sequence for two consecutive tokens under single-token decode is:

Step	**Token**	**Expert loaded**
1	t	$\mathcal{E}^{(0)}_{\sigma_t^{(0)}}$
2	t	$\mathcal{E}^{(1)}_{\sigma_t^{(1)}}$
3	t	$\mathcal{E}^{(2)}_{\sigma_t^{(2)}}$
4	t	$\mathcal{E}^{(3)}_{\sigma_t^{(3)}}$
5	$t+1$	$\mathcal{E}^{(0)}_{\sigma_{t+1}^{(0)}}$
6	$t+1$	$\mathcal{E}^{(1)}_{\sigma_{t+1}^{(1)}}$
$\vdots$	$\vdots$	$\vdots$

where $\sigma_t^{(\ell)}$ denotes the expert index selected for token t at layer ℓ. Each step requires loading a potentially different expert weight matrix. Under this access pattern, the token-level consistency loss reduces misses at step 5 relative to step 1 (same layer, adjacent tokens) but does nothing to reduce misses at step 2 relative to step 1 (adjacent layers, same token).

G.2 Cross-Layer Consistency and the Parameter Sharing Requirement

A natural extension is a *cross-layer consistency loss* that encourages the same expert to be selected at consecutive layers for the same token:

$$\mathcal{L}_{\text{cross}} = \frac{1}{T} \sum_{t=1}^{T} \sum_{\ell=1}^{L-1} \left\| \mathbf{g}_t^{(\ell)} - \mathbf{g}_t^{(\ell+1)} \right\|_2^2. \tag{18}$$

However, there is a critical prerequisite: cross-layer routing consistency only reduces cache misses if the *same expert index at different layers refers to the same weight tensor in memory*. In a standard MoE model where each layer maintains its own independent set of expert matrices $\{\mathcal{E}_i^{(\ell)}\}_{i=1}^N$, expert 2 at layer ℓ and expert 2 at layer $\ell+1$ are entirely different weight tensors. Routing consistently to expert 2 across both layers still requires two separate memory loads — no cache benefit is obtained.

Cross-layer consistency therefore becomes practically useful only under one of the following architectural conditions:

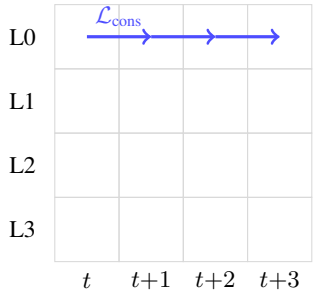
1. **Full cross-layer weight sharing.** Expert i is the same weight matrix at every layer: $\mathcal{E}_i^{(\ell)} = \mathcal{E}_i$ for all ℓ. Consistent routing to expert i across all layers requires only a single load, amortised over L layers of computation.
2. **Grouped weight sharing.** Layers are partitioned into blocks of size K (e.g. $K = 2$ or $K = 4$), and all layers within a block share the same expert pool. Consistent routing within a block reduces loads; routing across block boundaries still requires new loads.
3. **Universal expert pool.** All layers draw from one shared set of expert matrices — a design explored in recent parameter-efficient architectures. Any cross-layer routing consistency directly translates to cache reuse.

G.3 The Combined Objective

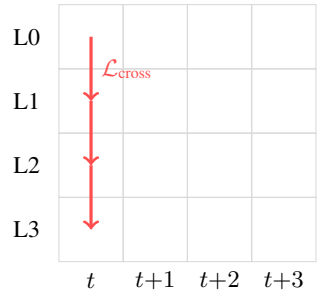
Under a grouped weight sharing architecture with block size K, the full memory-aware training objective combining token-level and cross-layer consistency would be:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda\,\mathcal{L}_{\text{cons}} + \mu\,\mathcal{L}_{\text{bal}} + \gamma\,\mathcal{L}_{\text{cross}}, \tag{19}$$

where $\gamma \geq 0$ controls the cross-layer consistency weight. The two consistency terms are complementary and target different bottlenecks: $\mathcal{L}_{\text{cons}}$ reduces misses across tokens at the same layer (prefill and batched decode), while $\mathcal{L}_{\text{cross}}$ reduces misses across layers for the same token (single-token decode). Figure 3 illustrates the two access patterns and the corresponding consistency objectives.



(a) Token-level consistency (prefill / batched decode). Arrows show penalised pairs under $\mathcal{L}_{\text{cons}}$ at layer L0.



(b) Cross-layer consistency (single-token decode). Arrows show penalised pairs under $\mathcal{L}_{\text{cross}}$ for token t.

Figure 3: The two memory access patterns and corresponding consistency objectives. Blue arrows (left) show token-level consistency targeting prefill and batched decode. Red arrows (right) show cross-layer consistency targeting single-token autoregressive decode. Both objectives are complementary and can be combined in eq. (19).

G.4 Limiting Case: Pre-defined Expert Trajectories

The consistency objectives proposed in this paper nudge the router toward stable routing decisions but still allow it to adapt freely. A natural limiting case is to remove the router entirely and fix expert assignments before training begins — defining a deterministic *trajectory* $\sigma_t = (\sigma_t^{(0)}, \sigma_t^{(1)}, \ldots, \sigma_t^{(L-1)})$ through the network for every token t, where $\sigma_t^{(\ell)} \in \{1, \ldots, N\}$ is the expert index at layer ℓ.

The idea of fixed routing is not new: Roller et al. (2021) show that assigning tokens to experts via a deterministic hash function, with no learned router, produces surprisingly competitive models.

Our trajectory framing differs in two key respects. First, assignments are coordinated *across layers* to define a complete network path per token, rather than independent per-layer hashes. Second, trajectories are chosen to maximise cache locality — consecutive tokens are assigned the same trajectory, so the entire sequence of expert weight loads for token t is identical to that of token $t-1$, yielding zero cache misses within a coherent span.

Under this design the cache miss rate during single-token autoregressive decode is determined entirely by how often the trajectory changes between consecutive tokens — a quantity that is fully under the designer's control rather than an emergent property of learned routing. In the extreme case where all tokens in a document share the same trajectory, cache misses are eliminated entirely at the cost of routing flexibility. This represents the upper bound on locality achievable by any routing strategy, and serves as a useful reference point for evaluating how close learned consistency objectives like STICKYMOE come to the theoretical optimum.

Whether a model trained with fixed locality-maximising trajectories can match the quality of a model with learned routing is an open empirical question. The hash routing results of Roller et al. (2021) suggest it is plausible, at least at small to moderate scale. We leave a systematic investigation of this trade-off to future work.

G.5 IMPLICATIONS AND FUTURE WORK

The analysis above identifies a gap in the current STICKYMOE formulation: it is most directly beneficial for prefill and batched decode, while single-token autoregressive decode — the dominant mode on memory-constrained edge devices — is better served by cross-layer consistency under a parameter-sharing architecture. Section 6.4 reports an empirical validation of $\mathcal{L}_{\text{cross}}$ under a universal (fully shared) expert pool: the cross-layer switch rate falls by 90–93%, confirming the mechanism proposed here is trainable and effective. What remains open is isolating the loss's marginal contribution from that of the shared-pool architecture itself (a $\gamma{=}0$ shared-pool ablation), and the interaction with the coarser *grouped* weight sharing described above, which trades off some of the memory savings of a universal pool for more per-layer routing flexibility. The training-time intervention principle extends naturally to this setting: just as token-level locality must be instilled during training rather than retrofitted, cross-layer locality requires architectural choices (parameter sharing) and training objectives ($\mathcal{L}_{\text{cross}}$) that are made from the outset.