TYING THE LOOP: TIED EXPERT LAYERS IN MIXTURE-OF-EXPERTS LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-Experts (MoE) architectures efficiently scale Large Language Models (LLMs) by activating only a small fraction of their experts per token, yet the full parameter count—dominated by the expert parameters—must be held in training and inference memory. To address this, we systematically study expert tying, an architectural modification that shares expert parameters across consecutive transformer layers while preserving independent, layer-wise routing and attention. We evaluate this approach across common, state-of-the-art architectures, including OLMoE, Qwen3, and DeepSeek-style MoEs. On full-size OLMoE-1B-7B, tying experts halves unique weight storage with virtually unchanged validation loss and similar downstream accuracy after 15.7B training tokens. Controlled ablations identify the value of preserving layer-specific attention under expert sharing. Reinvesting saved parameters in larger shared expert pools lowers validation loss at the baseline parameter budget across all configurations, making expert tying useful for both reducing storage and improving parameter allocation, and providing a more favorable compute-to-memory trade-off.

1 INTRODUCTION

Mixture-of-Experts (MoE) architectures have become a standard technique for scaling language models: by activating only a small subset of expert feed-forward networks (FFNs) per token, they decouple total parameter count from per-token compute (Shazeer et al., 2017; Fedus et al., 2022). Recent open-weights MoE models push this decoupling to extremes. DeepSeek-V3 activates only 37B of its 671B parameters per token ($\approx 5.5\%$), Qwen3-235B-A22B $\approx 9.4\%$, and Kimi-K2 $\approx 3.2\%$. As this active fraction shrinks, the memory footprint of an MoE is governed almost entirely by parameters that sit idle in any given forward pass: the full model must reside in training and inference memory even though only a tiny percentage of parameters contribute compute for each token.

This is in strong tension with a second trend: reasoning models and looped-depth models aim to extract more compute from each unique parameter, building more capable models at the same parameter count—the kind of parameter efficiency recently incentivized by OpenAI's Parameter Golf challenge (OpenAI, 2026). Together, these trends raise a design question: how much of MoE's expert capacity must be unique to each depth, and how much can be reused while preserving useful depth-specific computation?

Our approach. We study *expert tying* to reconcile the two opposing trends of the compute vs. memory trade-off. By reusing the same expert FFN weights across a group of consecutive layers while keeping routers, attention, and normalization layer-specific, we preserve MoE's low per-token compute yet raise compute per unique parameter, reducing the weight-storage cost of sparsity. Concretely, given a group of g layers, the gate/up/down projections of the N experts are aliased across all g layers, reducing the unique FFN parameters by a factor of g; each layer still computes its own routing distribution and its own attention output, so the hidden state continues to flow through g distinct layer operators, not g copies of the same one. The implementation is simple—a single Python-level pointer assignment in HuggingFace transformers models—and requires no changes to training or inference infrastructure beyond the optimizer, which must correctly accumulate gradients from the tied parameters' multiple use sites.

Why this works. Expert sharing need not make successive layers compute the same transformation. Each layer's attention mixes information across token positions before the shared experts act

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.

on the resulting hidden states. A common expert pool can therefore reuse feature transformations across depth while layer-specific attention changes the context in which they are applied. This motivates retaining per-layer attention rather than looping the entire transformer block. Our component ablation (Section 3) finds that, with experts shared, retaining per-layer attention reduces loss substantially; untying routers when both experts and attention are shared changes routing agreement much more than loss. This conditional result motivates sharing the largest parameter pool while retaining per-layer attention. In the reduced-scale configurations, $g = 4$ reduces total parameters by 43–52% at a loss cost of 0.048–0.067; in the 7B experiment the loss gap is 0.005 after 15.7B tokens.

Relation to looped transformers. Expert tying is a *partial loop*: expert FFNs are reused across depth while attention and routing remain layer-specific. Our contribution is a controlled study of which components can be shared at little quality cost and how the saved parameters can be reinvested in expert width. Section 2 situates this study within prior and parallel work.

Contributions. This paper makes the following contributions:

1. **Expert tying reduces storage at small quality cost.** Our experiments cover three MoE-style configurations and a full-size OLMoE-1B-7B model. Group-of-four tying reduces unique parameters by 43–52%; the 7B experiment reaches similar downstream scores at a 0.005 loss cost and yields 23.7% higher end-to-end training throughput with a larger microbatch in our DDP setup.
2. **Layer-specific attention matters when experts are shared.** We performed a controlled component ablation, separating the loss benefit of untying attention from the change in routing agreement produced by untying routers. Independent reruns reproduce a 0.040–0.047 loss penalty from additionally tying attention. A dense FFN-sharing control provides a reference for ordinary cross-depth FFN reuse.
3. **Heterogeneous width expansion** reinvests the parameters saved by expert tying into more experts in the tied middle layers, lowering validation loss by 0.027–0.030 at approximately the baseline parameter count in all three tested configurations. This recasts expert tying as a depth-vs-width design axis rather than just a compression technique.
4. We show that expert tying **composes cleanly with standard MoE training** recipes (load balancing, Muon/AdamW optimization), and requires no architectural changes to attention or routing, allowing expert reuse within existing MoE codebases.

2 Related Work

We focus on the closest architectural connections here; Appendix A provides a more extensive discussion.

Sparse MoE architectures. MoEs decouple total parameters from per-token compute by activating a subset of expert FFNs (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). Recent models combine large expert pools with fine-grained experts and improved routing and balancing (DeepSeek-AI, 2024; Yang et al., 2025; Muennighoff et al., 2024). ReMoE (Wang et al., 2025) and Expert Choice (Zhou et al., 2022) modify how tokens and experts are assigned. These advances largely retain layer-private expert pools. Expert tying instead reduces unique expert storage through cross-layer reuse while preserving the number and size of expert evaluations per token.

Looped transformers and iteration-specific computation. Universal Transformers (Dehghani et al., 2019) reuse transformer blocks across depth. Theory establishes the expressive power of looping and its usefulness for reasoning without proportional parameter growth (Giannou et al., 2023; Saunshi et al., 2025). Huginn (Geiping et al., 2025) and Ouro (Zhu et al., 2025) scale recurrent-depth pretraining, while Parcae (Prairie et al., 2026) and Schwethelm et al. (2026) study its scaling and efficiency trade-offs. Full-block sharing can restrict depth-specific computation, motivating per-iteration adaptation through low-rank modules (Bae et al., 2025a; Heo et al., 2025), explicit memory (Yu et al., 2025), or hyper-connections (Zeitoun et al., 2026). Other designs vary recurrence depth (Bae et al., 2025b; Jeddi et al., 2026), use routed LoRA modules (Nouriborji et al., 2025), or vary sequence resolution (Yu et al., 2026). Our approach is a partial loop: expert FFNs are shared, while attention, routers, and normalization remain layer-specific. The component ablation asks which of these freedoms materially improves quality, rather than assuming that each requires a separate adaptation mechanism.

Grouped expert sharing. Megrez2 (Li et al., 2025) is architecturally close: groups of consecutive layers share expert weights while retaining separate attention, gating, and normalization. Its production system combines this with pre-gated routing for memory-efficient inference. In closely related parallel work, MoRE (Qiu et al., 2026) shares enlarged expert pools across adjacent layers, retains separate attention and routers, and adds depth embeddings to router and expert inputs. Its parameter-matched pool expansion closely relates to our width-expansion experiments. Our contribution is a controlled study of the marginal effects of attention, router, and expert sharing, alongside compression and parameter-reinvestment settings across architectures and expert granularities. Thus, the contribution concerns the behavior and trade-offs of expert tying rather than the sharing configuration alone.

Full-block and global sharing. WideNet (Xue et al., 2022) shares attention, experts, and routers while retaining layer-specific normalization. MoEUT (Csordás et al., 2024) recurrently reuses MoE transformer blocks, including attention and routers, with modified gating and normalization. Loop-MoE (Chen et al., 2026a) adds iteration-conditioned normalization to shared blocks. Lee et al. (2026) associate fully looped MoE models' scaling gains over dense baselines with divergent expert selections across iterations. Their analysis concerns routing decisions under shared parameters; our ablation concerns the benefit of untying parameters. We find that untying routers changes routing agreement much more than loss, while untying attention yields larger loss gains. This distinguishes routing diversity from the marginal value of layer-specific router weights.

Global-pool approaches reuse experts throughout the network. MoUE (Chen et al., 2026b) combines a universal pool with structured expert exposure, reuse-aware balancing, and trajectory-aware routing. Hyper Experts (Deriy & Cetin, 2026) and UniPool (Huang et al., 2026) retain layer-specific routers; UniPool additionally uses pool-level balancing and NormRouter. CS-MoE (Jiao et al., 2026) combines layer-specific experts with a globally shared pool. These designs investigate different allocations of expert capacity across depth. Our principal setting shares experts within contiguous middle-layer groups, retains an untied prelude and coda, and uses standard MoE routing and losses. Varying group size and pool width exposes the trade-off between unique parameter count and cross-depth reuse.

Cross-layer routing and expert access. ReXMoE (Tan et al., 2025) broadens each layer's candidate pool to include neighboring experts, using progressive routing during training. PathMoE (Gu et al., 2026) shares router parameters across layers while keeping experts distinct, and Hu et al. (2026) coordinate adjacent routing decisions through an auxiliary coupling loss. These approaches alter routing structure or expert access; expert tying directly reduces unique expert tensors. They also motivate separating routing behavior from model quality: different routes through a shared pool need not require independent router parameters, and greater routing diversity alone need not imply lower loss. Our controlled comparison quantifies these distinctions under a common training setup.

3 Which Components Should Be Tied Across Layers?

A standard MoE transformer layer comprises four learned components: the expert FFN weights, the self-attention Q,K,V,O projections, the router, and the normalization layer gains (the RMSNorm scaling parameters). Each can in principle be tied (or "looped") across a group of consecutive layers or kept independent. Before turning to experiments on established MoE architectures (Section 4), we establish on a smaller controlled architecture which of these components must remain unique-per-layer to preserve quality, and which can share weights cheaply.

Setup. A depth-32 decoder-only transformer with $d_{\text{model}} = 512$, $n_{\text{heads}} = 8$, sequence length 512, and a fine-grained MoE FFN (32 experts, top-k $= 8$, $d_{\text{ff,expert}} = 256$). All blocks are pre-norm (Xiong et al., 2020) with RMSNorm (Zhang & Sennrich, 2019), rotary position embeddings (RoPE) (Su et al., 2024), and QK-norm (Henry et al., 2020) applied to queries and keys before the dot product. Full architectural and optimisation details are in Appendix B. We sweep three orthogonal axes:

- **Tying mode.** The three modes ALL-TIE, ATTN-TIE, and EXPERT-TIE differ in which of the four layer components are tied within a group; they form a monotone chain from most to least tied (Table 1).

- **Topology.** The 28 middle layers form k tying groups, each consisting of a parameter block of b unique transformer layers reused ℓ times in succession (so $k \cdot b \cdot \ell = 28$); we denote this $k\,(\text{group of } b)^{\ell\times}$. The 2-layer prelude and 2-layer coda are always left untied. We sweep five topologies: $1\,(\text{group of } 2)^{14\times}$, $2\,(\text{group of } 2)^{7\times}$, $3\,(\text{group of } 2)^{(5,4,5)\times}$ (non-uniform loop count, 5+4+5=14), $4\,(\text{group of } 1)^{7\times}$, and $7\,(\text{group of } 1)^{4\times}$. As an additional reference point we evaluate a universal transformer (MoEUT) (Csordás et al., 2024) configuration in which all 32 layers form a single tying group with no untied prelude or coda ($1\,(\text{group of } 2)^{16\times}$ over the full stack).
- **Granularity.** Fine (32 experts, top-8) versus coarse (8 experts, top-2, with per-expert FFN width scaled up so that active compute is unchanged).

All ablation runs use the same loss recipe as the production-architecture experiments in Section 4: a load-balancing auxiliary loss (Shazeer et al., 2017; Fedus et al., 2022) at coefficient $\alpha_{\text{aux}} = 10^{-2}$ and a router z-loss (Zoph et al., 2022) at coefficient $\alpha_z = 10^{-4}$, both added to the next-token cross-entropy.

Table 1: Tying modes. Within a tying group of g consecutive layers, each component is either *tied* (one parameter tensor shared by all g layers) or *per-layer* (each layer has its own copy). Norm gains are always per-layer in our codebase (each transformer layer has its own RMSNorm).

Mode	FFN experts	Attention	Router	Norm gains
ALL-TIE	tied	tied	tied	per-layer
ATTN-TIE	tied	tied	per-layer	per-layer
EXPERT-TIE	tied	per-layer	per-layer	per-layer

Routing diagnostic. Beyond final validation loss, we report *cross-loop agreement*: the average fraction of routing decisions that match across pairs of layers within the same tying group[1], measured on the held-out validation set at the end of training. A value of 1 means compared layers always select the same top-1 expert; a value of 0 means their top-1 choices never agree, without implying disjoint top-k selections. The metric measures routing differentiation, irrespective of whether the underlying expert weights or attention projections are also tied, and is well-defined for all three modes (in ALL-TIE mode, the same router still sees different hidden states at each layer, so agreement is non-trivial). Cross-loop agreement therefore lets us decouple the question "do tied layers *behave* like one operator?" from the question of model quality (loss).

3.1 Finding 1: Per-layer attention improves loss under expert sharing

Table 2 reveals a sharp decomposition. At fixed topology, untying *only* the router (ALL-TIE $\rightarrow$ ATTN-TIE) cuts cross-loop agreement by $\sim 4{-}7\times$ at every mid-stack topology ($0.733 \rightarrow 0.160$ at 2-group, $0.831 \rightarrow 0.126$ at 4-group), and by $\sim 10\times$ at MoEUT ($0.740 \rightarrow 0.071$). Untying *also* the attention (ATTN-TIE $\rightarrow$ EXPERT-TIE) produces only a smaller $\sim 1.1{-}1.8\times$ further drop. Per-layer routers do strongly individuate layers when free to, even though the underlying expert weights remain identical. The loss response is the inverse:

- Untying the router (ALL-TIE $\rightarrow$ ATTN-TIE) changes loss by at most 0.009 in any cell, and the sign is not consistent across topologies (ATTN-TIE better at 1- and 4-group; ALL-TIE marginally better at MoEUT and 2-group; tied at 3- and 7-group). Within the resolution of this ablation, the loss effect of router-untying is essentially zero.
- Untying attention (ATTN-TIE $\rightarrow$ EXPERT-TIE) reduces loss by 0.030–0.058 at every topology—several times the router effect and consistent in sign across topologies.

The distinction is useful for designing shared-expert models: visibly different expert choices do not by themselves imply better language modeling. In this ablation, freeing routers diversifies those choices without recovering the quality lost by sharing attention. Retaining per-layer attention delivers the larger, consistent loss improvement. Independent reruns at 2-group fine reproduce the

[1]For each token we compare the top-1 (`argmax`) expert selected by each layer in a tying group, and report the fraction of layer pairs that select the *same* top-1 expert, averaged over tokens and over the first 3 validation batches. The metric reflects top-1 routing only, irrespective of overlap among lower-ranked choices.

Table 2: Fine-grained ablation (32 experts, top-$k = 8$): final validation loss at 10,000 steps and cross-loop agreement. "Δ" is loss relative to the MoE baseline (no cross-layer tying). "Total" is unique parameter count; "Saved" is the relative reduction vs. the baseline. "MoEUT" denotes a single 32-layer tying group with no prelude or coda; all other rows have 28 middle layers in k tying groups with a 2+2 untied prelude/coda. Random chance cross-loop agreement (32 experts) is 0.031. Within each topology, modes are listed in order of increasing per-layer freedom.

Configuration	Total params	Saved ↑ params	Loss ↓	Δ ↓	Cross-loop agreement
Baseline (no cross-layer tying)					
MoE, no tying	488M	—	3.432	0.000	—
dense, no tying	186M	61.9%	3.560	+0.128	—
*MoEUT-style (*1 (*group of* 2)$^{16\times}$*, no prelude/coda)*					
MoEUT ALL-TIE	65M	86.7%	3.650	+0.218	0.740
MoEUT ATTN-TIE	66M	86.6%	3.653	+0.221	0.071
*Mid-stack (*28 *middle layers,* 2+2 *prelude/coda)*					
1 (group of 2)$^{14\times}$, ALL-TIE	121M	75.2%	3.553	+0.121	0.785
1 (group of 2)$^{14\times}$, ATTN-TIE	121M	75.1%	3.548	+0.116	0.155
1 (group of 2)$^{14\times}$, EXPERT-TIE	148M	69.6%	3.490	+0.058	0.087
2 (group of 2)$^{7\times}$, ALL-TIE	161M	67.1%	3.511	+0.079	0.733
2 (group of 2)$^{7\times}$, ATTN-TIE	161M	67.0%	3.512	+0.080	0.160
2 (group of 2)$^{7\times}$, EXPERT-TIE	186M	61.9%	3.469	+0.037	0.121
3 (group of 2)$^{(5,4,5)\times}$, ALL-TIE	151M	69.1%	3.481	+0.049	0.756
3 (group of 2)$^{(5,4,5)\times}$, ATTN-TIE	151M	69.0%	3.481	+0.049	0.193
3 (group of 2)$^{(5,4,5)\times}$, EXPERT-TIE	174M	64.5%	3.451	+0.019	0.107
4 (group of 1)$^{7\times}$, ALL-TIE	161M	67.1%	3.525	+0.093	0.831
4 (group of 1)$^{7\times}$, ATTN-TIE	161M	67.0%	3.516	+0.084	0.126
4 (group of 1)$^{7\times}$, EXPERT-TIE	186M	61.9%	3.470	+0.038	0.090
7 (group of 1)$^{4\times}$, ALL-TIE	202M	58.6%	3.485	+0.053	0.764
7 (group of 1)$^{4\times}$, ATTN-TIE	202M	58.5%	3.485	+0.053	0.142
7 (group of 1)$^{4\times}$, EXPERT-TIE	224M	54.1%	3.454	+0.022	0.126

additional attention-tying penalty (+0.03970 and +0.04662; Appendix D), supporting this finding beyond a single training run.

Component cost ranking. Reading the chain at 2-group fine yields Δ(tie experts) $\approx +0.037$, Δ(also tie attention) $\approx +0.043$, Δ(also tie router) ≈ 0 (slightly negative, -0.001), summing to the cumulative gap baseline $\rightarrow$ ALL-TIE of 0.079. The MoEUT-style configurations (which tie all components together) correspond to the regime measured at scale by Schwethelm et al. (2026); their finding that full-block recurrence carries substantial capacity cost is consistent with the gap we observe.

Two practical conclusions follow. First, expert-tying and additional attention-tying carry comparable incremental costs ($\approx +0.037$ and $\approx +0.043$ at 2-group fine), but the FFN expert pool dominates attention by an order of magnitude in parameter count, so expert-tying remains the highest-leverage memory move per parameter saved. Second, the router can be tied at no measurable cost on top of expert-and-attention-tying. The $+0.043$ attention figure is its cost *conditional on* experts already being tied; we did not run an attention-only-tied configuration. The ranking motivates our main design (Section 4): tie expert weights, keep attention and routers per-layer. Router sharing with per-layer attention is not isolated by this ablation.

Dense sharing control. Sharing dense FFNs while keeping attention per-layer reduces unique parameters by 40.7% at a loss cost of 0.0694 (Appendix D). In the corresponding fine-grained four-group MoE setting, expert sharing removes 61.9% of parameters at a loss cost of 0.038. Thus, expert sharing gives a more favorable relative compression trade-off here, although the baseline capacities differ. At approximately matched total capacity (186M unique parameters), the two-group expert-

Table 3: Coarse-grained variant (8 experts, top-$k = 2$, active FFN compute matched). The component ordering and the loss-vs-routing decoupling observed at fine granularity hold without modification. "Total" is unique parameter count; "Saved" is the relative reduction vs. the coarse baseline. Random chance cross-loop agreement (8 experts) is 0.125.

Configuration	Total params	Saved ↑ params	Loss ↓	Δ ↓	Cross-loop agreement
Baseline MoE (no cross-layer tying)	488M	—	3.481	0.000	—
MoEUT ALL-TIE (1 (group of 2)$^{16\times}$, full stack)	65M	86.7%	3.663	+0.182	0.896
MoEUT ATTN-TIE (1 (group of 2)$^{16\times}$, full stack)	65M	86.7%	3.664	+0.183	0.172
2 (group of 2)$^{7\times}$, EXPERT-TIE	186M	61.9%	3.489	+0.008	0.192
4 (group of 1)$^{7\times}$, ALL-TIE	161M	67.1%	3.559	+0.078	0.847
4 (group of 1)$^{7\times}$, ATTN-TIE	161M	67.0%	3.546	+0.065	0.226
4 (group of 1)$^{7\times}$, EXPERT-TIE	186M	61.9%	3.489	+0.008	0.185
7 (group of 1)$^{4\times}$, EXPERT-TIE	224M	54.2%	3.475	−0.006	0.185

tied MoE also reaches lower loss than the untied dense reference (3.469 vs 3.560; Table 2). Selected FFN width is matched, but these architectural comparisons do not isolate a causal benefit of routing.

3.2 Finding 2: Topology trades parameter savings against modest loss changes

Once the tying mode is fixed at EXPERT-TIE, partitioning the 28 middle layers into groups is a relatively minor knob: loss across $k \in \{2, 3, 4, 7\}$ groups varies by only 0.019 (3.451 to 3.470), and adding 1-group (the most aggressive sharing, at 148M unique params) extends the range to 0.039. Cross-loop agreement is similarly stable (range 0.087–0.126), confirming that topology choice does not qualitatively change routing behaviour. Topology within EXPERT-TIE therefore sets the parameter-savings ratio without large quality consequences—a knob we use in Section 4 to interpret tie-group size as a parameter-efficiency dial.

3.3 Finding 3: Component ranking holds at both tested expert granularities

A natural concern is that the dominance of attention over the router might depend on the routing capacity afforded by fine-grained MoE: with 32 experts and top-$k = 8$, each layer's router has rich combinatorial flexibility; with coarse routing (8 experts, top-2) it does not. We re-run the MoE baseline, MoEUT ALL-TIE, MoEUT ATTN-TIE, the three-mode 4-group cross-section, and EXPERT-TIE at two further topologies (2- and 7-group) at the coarse setting, with per-expert FFN width scaled up $4\times$ so that active FFN compute is exactly matched ($2 \times 1024 = 8 \times 256$). The pattern in Table 3 is the same as at fine expert granularity. The router-untying step (ALL-TIE → ATTN-TIE at 4-group) collapses cross-loop agreement by $3.7\times$ ($0.847 \rightarrow 0.226$) while moving loss by only 0.013. The attention-untying step (ATTN-TIE → EXPERT-TIE) produces a $1.22\times$ residual agreement drop and a 0.057 loss improvement. Topology spread within EXPERT-TIE remains small (0.014). The component ranking, the routing–loss decoupling, and the practical conclusion are robust to granularity.

3.4 Finding 4: Boundary placement has a small effect in a matched control

The MoEUT-versus-mid-stack comparisons also change unique parameter count and tying structure, so they do not isolate the effect of the prelude/coda. We therefore compare five placements of four untied layers at fixed parameters (186.1M), effective depth, active compute, repeat-count profile, and tied-LR treatment (Appendix D). Loss ranges from 3.4667 to 3.4734, with the 2+2 boundary placement at 3.4689. Each placement has one run, and most pairwise differences are comparable to the observed run-to-run variation. The control does not establish that 2+2 is necessary or optimal; we retain it as a conservative configuration rather than a principal source of the gains.

3.5 Optimization dynamics of tied experts

When expert parameters are shared across g consecutive layers, each shared tensor receives one update per step but acts g times in the forward pass, which suggests reducing its learning rate. The classical heuristic for shared weights (LeCun et al., 1998; Hoffer et al., 2017) prescribes $1/\sqrt{g}$ under a random-walk assumption on the summed gradient. That argument does not transfer directly to our optimisers: Muon's Newton–Schulz orthogonalisation (like Adam's per-coordinate normalisation) makes the update magnitude largely insensitive to the gradient's scale, so summing gradients from g use-sites changes the update's direction rather than its size. A more relevant consideration is *forward-pass* amplification: the same ΔW acts g times in sequence. If a token were routed to the same expert at every depth, this would motivate a linear $1/g$ penalty, analogous to the residual-stream growth observed in fully looped architectures (Prairie et al., 2026); with independent per-layer routers and intermediate self-attention, tokens take shifting paths and a milder penalty is plausible. These considerations motivate the candidate rules but do not determine the choice, which we make empirically.

Empirical validation. We ablate the divisor on the 4-group topology. Averaging the final validation loss across all three tying modes (ALL-TIE, ATTN-TIE, EXPERT-TIE), the $1/\sqrt{g}$ rule (3.503) and the strict $1/g$ rule (3.506) perform near-identically, while *no* scaling is clearly worst (3.542). The ablation thus supports reducing the tied-parameter learning rate, but the two scaled rules are indistinguishable at this resolution, so it does not by itself establish sub-linear amplification. We adopt $1/\sqrt{g}$ as the default for the remainder of this paper, as the milder of two empirically equivalent rules.

4 Main Experiments on MoE Architectures

Motivated by Section 3, we tie expert FFN weights and retain per-layer attention and routers, using a fixed $2{+}2$ untied prelude/coda. We apply this design to configurations based on OLMoE (Muennighoff et al., 2024), Qwen3-MoE (Yang et al., 2025), and DeepSeekMoE (DeepSeek-AI, 2024), and study two practical questions: (i) how much parameter saving does $g{=}4$ expert tying buy, and at what quality cost? (ii) does heterogeneous *width expansion*—reinvesting the parameters saved by tying as additional experts in the tied middle layers—recover quality at iso-parameter count, turning expert tying from compression into a depth-vs-width design axis?

Setup. For each architecture we train five configurations. The *baseline* ($g{=}1$) is the standard untied MoE; two *tied* configurations share expert weights across consecutive groups of $g{=}2$ and $g{=}4$ layers, with a $2{+}2$ untied prelude and coda; and two *width-expanded* configurations widen the tied middle layers (at $g{=}4$) to $2\times$ and $4\times$ the baseline expert count, with the $4\times$ variant chosen so that total parameter count returns to within 1% of the baseline ("iso-base"). Selected-expert FFN computation is unchanged within each architecture; larger expert pools slightly increase router computation. Here, "width" means candidate expert count; each expert's FFN dimension is fixed. An untied g-layer group stores gN experts; a tied group with $w\times$ expansion stores wN. Thus $w = g$ preserves expert storage, top-k, and effective depth while reallocating capacity into a shared candidate pool. All variants train on a $75{:}25$ mixture of DCLM-edu (Allal et al., 2025) and FinePhrase (Niklaus et al., 2026) for $20{,}000$ steps at effective batch size 524k tokens (≈ 10.5B tokens total). Models are scaled-down variants of the named production architectures; full hyperparameters in Appendix C.

Optimizer. We use Muon (Jordan et al., 2024; Liu et al., 2025) for the 2D hidden weights—attention Q,K,V,O projections and FFN gate/up/down projections—at base learning rate $\eta_{\text{Muon}} = 2 \times 10^{-2}$ with weight decay 0.1. AdamW (Loshchilov & Hutter, 2019) handles the embeddings, output head, norm gains, biases, and routers at $\eta_{\text{AdamW}} = 0.1 \cdot \eta_{\text{Muon}} = 2 \times 10^{-3}$ with weight decay 0.01.[2] Tied expert weight tensors receive a learning rate divided by $\sqrt{g}$, the rule selected empirically in Section 3.5. Weight decay is left uncompensated: tied and untied parameter groups use the same base λ (baseline $\lambda = 0.1$ on Muon, $\lambda = 0.01$ on AdamW; see Appendix C). Both optimisers follow the same cosine schedule (linear warmup, decay to $0.1\times$ peak). All configurations train with a load-balancing auxiliary loss (Shazeer et al., 2017; Fedus et al., 2022) at coefficient $\alpha_{\text{aux}} = 10^{-2}$ and

[2]Routers are kept on AdamW even though their weights are 2D, because the router output behaves as a per-token classifier head where adaptive per-parameter learning rates suit the heavy-tailed gradient distribution. Weight decay for AdamW parameters according to Jordan et al. (2024).

Table 4: Final pretraining and downstream metrics on three MoE architectures (native depths: 16 layers for OLMoE/DeepSeekMoE, 28 for Qwen3-MoE), 20,000 steps ($\approx$ 10.5B tokens). Per-architecture configurations: *baseline* = standard untied MoE; g=4 = expert weights tied across consecutive groups of 4 layers with 2+2 untied prelude/coda; $+\,w\times$ *width* = the tied middle layers widened to w times the baseline expert count. The $4\times$ row matches each baseline's total-parameter budget ("iso-base"). *Active* counts forward-pass FLOPs in parameter-equivalents; *Total* reflects physical (unique) parameter storage. If no expert is reused for a token, the computation-based and unique-parameter counts of the selected experts coincide. If the same expert is selected multiple times within a tied group, each layer's computation contributes to *Active*, although those weights are counted only once in *Total*. Selected-expert FFN computation is unchanged, while wider pools add router computation. *Saved* is the relative reduction in total parameters vs. that architecture's baseline. LM Loss on validation set, not including aux losses. *Avg Acc* is the macro-average 3-shot accuracy on {ARC-Easy, ARC-Challenge, HellaSwag, PIQA, WinoGrande, OpenBookQA} via `lm-evaluation-harness`, using plain `acc` for every task. Arrows mark direction of preference.

Arch	Topology	Active	Total	Saved ↑	Loss ↓	PPL ↓	Avg Acc% ↑
DeepSeekMoE	baseline	158M	523M	—	3.132	22.92	41.5
	g=2		372M	29%	3.149	23.30	41.2
	g=4		296M	43%	3.180	24.06	41.0
	g=4, 2× width		372M	29%	3.143	23.19	40.9
	g=4, 4× width *(iso-base)*		523M	≈ 0%	3.105	22.31	41.9
OLMoE	baseline	170M	523M	—	3.119	22.62	41.9
	g=2		372M	29%	3.136	23.00	42.1
	g=4		296M	43%	3.171	23.83	40.8
	g=4, 2× width		372M	29%	3.132	22.93	41.5
	g=4, 4× width *(iso-base)*		523M	≈ 0%	3.089	21.95	42.1
Qwen3-MoE	baseline	112M	459M	—	3.171	23.84	41.7
	g=2		300M	35%	3.201	24.55	40.6
	g=4		220M	52%	3.238	25.47	39.8
	g=4, 2× width		300M	35%	3.192	24.34	40.4
	g=4, 4× width *(iso-base)*		459M	≈ 0%	3.142	23.17	41.6

a router z-loss (Zoph et al., 2022) at coefficient $\alpha_z = 10^{-4}$, both added to the next-token cross-entropy. We extensively monitor router health and confirm that cross-layer tying does not induce expert collapse (see Appendix G for detailed routing dynamics). Full details and hyperparameters are given in Appendix C.

Expert tying trades parameters for a modest loss increase. Going from baseline to tied/looped configurations saves 29–52% of total parameters at a modest quality cost that scales monotonically with the tying group size. At g=2 the loss penalty is only 0.02–0.03 (OLMoE 3.119→3.136, DeepSeekMoE 3.132→3.149, Qwen3-MoE 3.171→3.201); at g=4 it grows to 0.05–0.07 (OLMoE +0.052, DeepSeekMoE +0.048, Qwen3-MoE +0.067) while saving 43–52% of total parameters. Average downstream accuracy follows the same broad trend, dropping by at most 1.9 percentage points at g=4. The cost is consistent across all three architectures and free of any cliff: the smooth baseline→ g=2 → g=4 progression shows that quality degrades gracefully as more layers share weights. The full-scale OLMoE experiment below has a smaller loss gap at its evaluated training horizon.

At the baseline parameter budget, width expansion lowers loss. Reinvesting saved parameters as additional experts in the tied middle layers (g=4, 4× width) lowers loss by 0.027–0.030 relative to the untied baselines (OLMoE 3.089 vs 3.119, DeepSeekMoE 3.105 vs 3.132, Qwen3-MoE 3.142 vs 3.171). Downstream differences are small and mixed. At full scale, where no iso-parameter variant was trained, partial reinvestment (g=4, 2× width) also lowers loss (2.812 vs 2.820) while retaining 33.8% fewer parameters (Table 5). The consistent loss gains show that the value of tying extends beyond compression: at the tested baseline budgets, allocating more candidate experts to a pool reused across depth improves quality without increasing selected-expert FFN computation. The best allocation remains dependent on the budget and training horizon.

Efficiency and throughput gains. Beyond memory savings, expert tying can accelerate wall-clock training. Fewer unique parameters reduce optimizer-state storage and update work, as well as gradient communication under data parallelism. The resulting memory headroom also permits larger microbatches. On $4\times$H200 GPUs with PyTorch DDP, full-scale $g{=}4$ OLMoE achieves $51{,}777$ versus $41{,}859$ tokens/sec ($+23.7\%$), using a larger microbatch (16 vs 4) at the same $524{,}288$-token global batch. This end-to-end gain includes the benefit of the larger microbatch. Reduced-scale OLMoE gains 15.7% at matched microbatch size, with reduced DDP communication contributing in both settings (Appendix C). At full scale, measured resident-weight memory falls 50.9% and peak inference memory falls 26.2–42.4% (one H200, float32; Appendix E).

Scaling to 7B model size. We train the full-size OLMoE-1B-7B architecture (7.12B total, ≈ 1.3B active; 16 layers, 2048 hidden, 64 experts, top-8) from scratch in three configurations: untied, $g{=}4$, and $g{=}4$ with $2\times$ width, each for $30{,}000$ steps ($\approx$ 15.7B tokens) on $4\times$H200 GPUs under the same recipe. Plain tying yields loss 2.825 versus 2.820 and downstream accuracy 47.5% versus 47.4%, while reducing unique parameters from 7.12B to 3.50B. Partial reinvestment (4.71B parameters) reaches loss 2.812 and accuracy 48.2%, while retaining one-third less weight storage than the untied baseline. Across training, plain $g{=}4$ closes its early loss gap by $10{,}000$ steps, while the $2\times$-expanded model has lower validation loss than the untied baseline at all 50 logged evaluations after 2.6B tokens. This consistency is within one training run per configuration; the small downstream gaps indicate similar observed performance (additional evaluation in Appendix F). As an external reference, our $g{=}4$ checkpoint at 10.5B tokens scores 45.1% on the same six-task protocol, versus 44.4% for the official OLMoE-1B-7B-0924 `step5000-tokens20B` checkpoint at approximately 20B tokens. This is a recipe sanity check: the tokenizer, training data, and optimizer differ, so the comparison does not isolate the effect of tying.

Table 5: Full-scale OLMoE-1B-7B at $30{,}000$ steps ($\approx$ 15.7B tokens). Columns and six-task plain-`acc` protocol follow Table 4. Group-of-four tying halves unique parameter storage with similar observed downstream accuracy. The official OLMoE checkpoint is an external reference with a different training recipe; our $g{=}4$ checkpoint already scores 45.1% at 10.5B tokens versus its 44.4% at approximately 20B tokens. OLMoE-1B-7B-0924 has the same architecture and parameter count as our untied baseline apart from the vocabulary-dependent embedding sizes: the larger `cl100k` vocabulary used in all our runs gives our baseline 7.12B parameters versus the reference's 6.9B.

Configuration	Total	Saved ↑	Loss ↓	PPL ↓	Avg Acc% ↑
baseline ($g{=}1$)	7.12B	—	2.820	16.78	47.4
$g{=}4$	3.50B	50.9%	2.825	16.86	47.5
$g{=}4$, $2\times$ width	4.71B	33.8%	2.812	16.64	48.2
OLMoE-1B-7B-0924 (ref., $\approx 20B$ tok)	6.9B	—	N/A	N/A	44.4

5 Conclusion

Our results support a practical design choice for MoE models: share expert pools across nearby layers, retain layer-specific attention, and choose the pool size according to the parameter budget. The component ablation and independent reruns establish the value of preserving attention under expert sharing. Plain tying reduces weight storage at small loss cost; reinvesting the saving in more candidate experts lowers loss at the tested baseline budgets without increasing selected-expert FFN computation. Together with measured inference-memory and DDP throughput benefits, these findings support expert tying as both a storage-reduction technique and a way to improve parameter allocation.

Limitations. Full-scale evidence covers one architecture family and 15.7B training tokens; longer-horizon parity is untested. The main comparisons use one training run per configuration, and evaluation-set uncertainty does not capture training variance. Coding, long-context, and instruction-following behavior are unevaluated. Systems measurements are specific to H200 hardware and DDP training; training peak memory, inference latency, and EP/TP scaling are unmeasured.

REPRODUCIBILITY STATEMENT

Architecture, optimization, data, hardware, and evaluation details are given in Appendix B and Appendix C.

AI USE STATEMENT

We used AI tools to assist with code development and to refine the drafting and phrasing of the manuscript. All assisted code and text parts were reviewed by the authors, who take full responsibility for the paper's content and results.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM2: When smol goes big – data-centric training of a small language model, 2025. URL https://arxiv.org/abs/2502.02737. Dataset card: https://huggingface.co/datasets/HuggingFaceTB/dclm-edu.

Sangmin Bae, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Seungyeon Kim, and Tal Schuster. Relaxed recursive transformers: Effective parameter sharing with layer-wise LoRA. In *International Conference on Learning Representations (ICLR)*, 2025a. URL https://arxiv.org/abs/2410.20672.

Sangmin Bae, Yujin Kim, Reza Bayat, Sungnyun Kim, Jiyoun Ha, Tal Schuster, Adam Fisch, Hrayr Harutyunyan, Ziwei Ji, Aaron Courville, and Se-Young Yun. Mixture-of-recursions: Learning dynamic recursive depths for adaptive token-level computation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025b. URL https://arxiv.org/abs/2507.10524.

Wenkai Chen, Tianshu Li, Wenyong Huang, Yichun Yin, Lifeng Shang, and Chengwei Qin. Loopmoe: Unifying iterative computation with mixture-of-experts for language modeling. *arXiv preprint arXiv:2606.04438*, 2026a.

Yilong Chen, Naibin Gu, Junyuan Shang, Zhenyu Zhang, Yuchen Feng, Jiawei Sheng, Tingwen Liu, Shuohuan Wang, Yu Sun, Hua Wu, and Haifeng Wang. Mixture of universal experts: Scaling virtual width via depth-width transformation. *arXiv preprint arXiv:2603.04971*, 2026b.

Róbert Csordás, Kazuki Irie, and Jürgen Schmidhuber. MoEUT: Mixture-of-experts universal transformers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. URL https://arxiv.org/abs/2405.16039.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. Universal transformers. In *International Conference on Learning Representations (ICLR)*, 2019. URL https://arxiv.org/abs/1807.03819.

Ilya Deriy and Edoardo Cetin. Hyper experts: Language models with inference-time layer reallocation. In *ICLR 2026 Workshop on Test-Time Updates*, 2026. URL https://openreview.net/forum?id=cGD7Nzuihz.

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.

Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. Scaling up test-time compute with latent reasoning: A recurrent depth approach. *arXiv preprint arXiv:2502.05171*, 2025.

Angeliki Giannou, Shashank Rajput, Jy-Yong Sohn, Kangwook Lee, Jason D. Lee, and Dimitris Papailiopoulos. Looped transformers as programmable computers. In *International Conference on Machine Learning (ICML)*, pp. 11398–11442, 2023.

Zijin Gu, Tatiana Likhomanenko, Vimal Thilak, Jason Ramapuram, and Navdeep Jaitly. Path-constrained mixture-of-experts. *arXiv preprint arXiv:2603.18297*, 2026.

Alex Henry, Prudhvi Raj Dachapally, Shubham Shantaram Pawar, and Yuxuan Chen. Query-key normalization for transformers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 4246–4253. Association for Computational Linguistics, 2020. doi: 10.18653/v1/2020.findings-emnlp.379. URL https://aclanthology.org/2020.findings-emnlp.379/.

Jaemu Heo, Eldor Fozilov, Hyunmin Song, and Taehwan Kim. RingFormer: Rethinking recurrent transformer with adaptive level signals. *arXiv preprint arXiv:2502.13181*, 2025.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: Closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 1731–1741, 2017. URL https://papers.nips.cc/paper_files/paper/2017/hash/a5e0ff62be0b08456fc7f1e88812af3d-Abstract.html.

Rizhen Hu, Yuan Cao, Boao Kong, Mou Sun, and Kun Yuan. Synergistic intra-and cross-layer regularization losses for moe expert specialization. *arXiv preprint arXiv:2602.14159*, 2026.

Minbin Huang, Han Shi, Chuanyang Zheng, Yimeng Wu, Guoxuan Chen, Xintong Yu, Yichun Yin, and Hong Cheng. UniPool: A globally shared expert pool for mixture-of-experts. *arXiv preprint arXiv:2605.06665*, 2026. URL https://arxiv.org/abs/2605.06665.

Ahmadreza Jeddi, Marco Ciccone, and Babak Taati. LoopFormer: Elastic-depth looped transformers for latent reasoning via shortcut modulation. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://arxiv.org/abs/2602.11451.

Dian Jiao, Jiaxin Duan, Shuai Zhao, Jiabing Leng, Yiran Zhang, and Feng Huang. Improving parameter utilization by sharing neural experts across layers in transformers. *arXiv preprint arXiv:2609.22199*, 2026. URL https://arxiv.org/abs/2609.22199.

Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient Backprop. In Genevieve B. Orr and Klaus-Robert Müller (eds.), *Neural Networks: Tricks of the Trade*, volume 1524 of *Lecture Notes in Computer Science*, pp. 9–50. Springer, 1998. doi: 10.1007/3-540-49430-8_2.

Ryan Lee, Jacob Biloki, Edward J. Hu, and Jonathan May. Sparse layers are critical to scaling looped language models. *arXiv preprint arXiv:2605.09165*, 2026. URL https://arxiv.org/abs/2605.09165.

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. URL https://arxiv.org/abs/2006.16668.

Boxun Li, Yadong Li, Zhiyuan Li, Congyi Liu, Weilin Liu, Guowei Niu, Zheyue Tan, Haiyang Xu, Zhuyu Yao, Tao Yuan, Dong Zhou, Yueqing Zhuang, Bo Zhao, Guohao Dai, and Yu Wang. Megrez2 technical report. *arXiv preprint arXiv:2507.17728*, 2025.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for LLM training, 2025. URL https://arxiv.org/abs/2502.16982.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, et al. OLMoE: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*, 2024.

Joel Niklaus, Guilherme Penedo, Hynek Kydlicek, Elie Bakouch, Lewis Tunstall, Ed Beeching, Thibaud Frere, Colin Raffel, Leandro von Werra, and Thomas Wolf. The synthetic data playbook: Generating trillions of the finest tokens. https://huggingface.co/spaces/HuggingFaceFW/finephrase, 2026. HuggingFace Space and dataset, https://huggingface.co/datasets/HuggingFaceFW/finephrase.

Mohammadmahdi Nouriborji, Morteza Rohanian, and Omid Rohanian. Improving recursive transformers with mixture of LoRAs. *arXiv preprint arXiv:2512.12880*, 2025.

OpenAI. What parameter golf taught us. https://openai.com/index/what-parameter-golf-taught-us/, 2026. Accessed: 2026-06-14.

Hayden Prairie, Zachary Novack, Taylor Berg-Kirkpatrick, and Daniel Y. Fu. Parcae: Scaling laws for stable looped language models. *arXiv preprint arXiv:2604.12946*, 2026.

Eric S. Qiu, Utku Umur Acikalin, Justin Lovelace, Christian Belardi, Arjun B. Mulchandani, Carla P. Gomes, and Kilian Q. Weinberger. MoRE: Mixture of reused experts. In *Conference on Language Modeling*, 2026. URL https://arxiv.org/abs/2609.18176.

Nikunj Saunshi, Nishanth Dikkala, Zhiyuan Li, Sanjiv Kumar, and Sashank J. Reddi. Reasoning with latent thoughts: On the power of looped transformers. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2502.17416.

Kristian Schwethelm, Daniel Rueckert, and Georgios Kaissis. How much is one recurrence worth? iso-depth scaling laws for looped language models. *arXiv preprint arXiv:2604.21106*, 2026.

Noam Shazeer. GLU variants improve transformer. 2020.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1701.06538.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024.

Zheyue Tan et al. ReXMoE: Reusing experts with minimal overhead in mixture-of-experts. *arXiv preprint arXiv:2510.17483*, 2025.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

Ziteng Wang, Jianfei Chen, and Jun Zhu. ReMoE: Fully differentiable mixture-of-experts with ReLU routing. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2412.14711.

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Chen Xing, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tieyan Liu. On layer normalization in the transformer architecture. In *International Conference on Machine Learning (ICML)*, pp. 10524–10533. PMLR, 2020.

Fuzhao Xue, Ziji Shi, Futao Wei, Yuxuan Lou, Yong Liu, and Yang You. Go wider instead of deeper. In *AAAI Conference on Artificial Intelligence*, 2022.

An Yang, Baosong Yang, Beichen Zhang, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

Chengting Yu et al. MeSH: Memory-as-state-highways for recursive transformers. *arXiv preprint arXiv:2510.07739*, 2025.

Chengting Yu et al. SpiralFormer: Looped transformers can learn hierarchical dependencies via multi-resolution recursion. *arXiv preprint arXiv:2602.11698*, 2026.

Abbas Zeitoun, Lucas Torroba-Hennigen, and Yoon Kim. Hyperloop transformers. *arXiv preprint arXiv:2604.21254*, 2026.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 32, 2019.

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

Rui-Jie Zhu, Zixuan Wang, Kai Hua, Tianyu Zhang, Ziniu Li, Haoran Que, Boyi Wei, Zixin Wen, Fan Yin, et al. Scaling latent reasoning via looped language models. *arXiv preprint arXiv:2510.25741*, 2025.

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. *arXiv preprint arXiv:2202.08906*, 2022.

A Extended Related Work

A.1 Mixture-of-Experts Architectures

Mixture-of-Experts (MoE) architectures decouple total parameter count from per-token compute by activating only a subset of expert FFN networks per token (Shazeer et al., 2017; Lepikhin et al., 2021; Fedus et al., 2022). Recent work has pushed toward *fine-grained* experts with large expert pools: DeepSeek-V3 (DeepSeek-AI, 2024) uses 256 routed experts with an auxiliary-loss-free load-balancing strategy; Qwen3 (Yang et al., 2025) employs 128 experts; Kimi-K2 scales to 384. These models demonstrate that richer expert combinations with smaller individual experts consistently improve expressiveness and downstream quality. OLMoE (Muennighoff et al., 2024) shows that even at the 7B-total / 1B-active scale, MoE can outperform dense models trained with 6–7$\times$ more compute. Routing mechanisms remain an active research area: ReMoE (Wang et al., 2025) replaces top-k + softmax routing with continuous ReLU routing for full differentiability, and Expert Choice routing (Zhou et al., 2022) inverts the standard token-to-expert assignment.

Despite strong scaling properties, MoE models carry a substantial memory footprint due to the large number of expert parameters. While only a small fraction of the parameters are activated per token, the full parameters still need to reside in memory. This motivates cross-layer expert sharing as a means to reduce the unique parameter count without changing the activated compute budget.

A.2 Looped Transformers and Parameter Sharing

Our work is closely connected to a growing body of research on *looped transformers*—architectures that reuse the same block of layers multiple times, increasing effective depth without proportionally increasing parameter count. Under this lens, our approach is a partial loop: the MoE FFN sub-block is shared ("looped") across a group of consecutive layers, while attention, normalization, and routers remain distinct at each position in the group.

Foundations and theory. Universal Transformers (Dehghani et al., 2019) first demonstrated that sharing a single transformer block across depth can match standard transformers on certain tasks via adaptive-computation-time halting. Giannou et al. (2023) provide a constructive expressivity result, showing that a constant-depth transformer placed in a loop can emulate a programmable instruction-set computer. Saunshi et al. (2025) formalized the reasoning inductive bias of this design, showing both theoretically and empirically that a k-layer transformer looped L times closely matches the performance of a kL-layer non-looped model on reasoning-intensive tasks, despite using $L\times$ fewer unique parameters. Their central claim—that many reasoning problems require depth but not necessarily parameters—provides the theoretical basis for the broader looped transformer program.

Scaling looped pretraining. Recent work has demonstrated that looped architectures can be scaled effectively. *Huginn* (Geiping et al., 2025) introduces a depth-recurrent 3.5B-parameter LM with a prelude–core–coda topology, trained on 800B tokens, where unrolling the core block more times at inference improves downstream quality. *Ouro* (Zhu et al., 2025) pretrains 1.4B and 2.6B looped models for 7.7T tokens, matching the performance of 4B and 8B standard transformers respectively. Ouro's controlled experiments offer a key insight for our work: looping does not increase knowledge *capacity* per parameter (both looped and non-looped models encode roughly 2 bits/parameter on synthetic memorization tasks), but it preserves or improves knowledge *manipulation* on multi-hop reasoning. *Parcae* (Prairie et al., 2026) establishes the first scaling laws for looped architectures, showing that a 770M Parcae matches a 1.3B parameter transformer on the same data, and that compute-optimal training requires scaling loop count and data together. Parcae also identifies stability bottlenecks (residual explosion, loss spikes) specific to loop-based training and addresses them via spectral-norm constraints on the input injection. Most recently, Schwethelm et al. (2026) fit scaling laws for fully-looped language models and find that each additional recurrence contributes substantially less than an additional unique block, providing a quantitative complement to our component-ablation perspective.

Differentiating iterations. A central challenge in fully-looped architectures is that the same block processes every iteration, limiting per-depth specialization. Several methods add lightweight per-iteration adaptation on top of a shared core. *Relaxed Recursive Transformers* (Bae et al., 2025a) attach depth-wise LoRA modules to each loop iteration of a dense recursive transformer, recovering most of the performance of the original full-size model when retrofitting pretrained LLMs

into looped form. *RingFormer* (Heo et al., 2025) generates input-dependent level signals via depth-specific low-rank matrices that modulate the shared block at each iteration. *LoopFormer* (Jeddi et al., 2026) trains a looped transformer on variable unrolling depths with a consistency loss between short and long trajectories. *MeSH* (Yu et al., 2025) externalizes state management into an explicit memory buffer with step-wise routers to diversify computation across iterations. *Mixture-of-Recursions* (Bae et al., 2025b) uses lightweight routers to assign different recursion depths to individual tokens, unifying parameter sharing with adaptive computation. *ModernALBERT / Mixture of LoRAs* (Nouriborji et al., 2025) inserts LoRA experts with token-conditional routing inside the shared FFN of a recursive encoder. *SpiralFormer* (Yu et al., 2026) introduces multi-resolution recursion, operating the shared core at varying sequence resolutions across loop iterations. All of these works operate on dense (non-MoE) transformers. Concurrently, *Hyperloop Transformers* (Zeitoun et al., 2026) loop a middle block (with untied begin/end layers) in dense transformers, adding loop-level hyper-connections to differentiate iterations. Their dense design adds an explicit per-iteration mechanism; our component ablation instead measures the benefit of retaining per-layer attention when experts are tied.

A.3 Cross-Layer Sharing in MoEs

The idea of sharing expert parameters across layers in MoE architectures has emerged in several recent works. Because the FFN/expert block carries the bulk of parameters in a modern MoE, sharing experts is the most impactful axis of parameter reduction—and, through the lens of looping, corresponds to looping only the MoE sub-block while leaving attention and routing free to vary per layer.

MoEUT (Csordás et al., 2024) combines fully-shared MoE layers in a Universal Transformer framework. MoEUT uses σ-MoE for both attention and feedforward layers, groups non-shared layers into blocks that are then recurrently stacked, and introduces a custom "peri-layernorm" scheme. MoEUT shares every component—including routers and attention—across loop iterations, and does not explore which components to untie.

Megrez2 (Li et al., 2025) is architecturally close to our work: it partitions an L-layer transformer into groups of n consecutive layers that share expert weights while retaining per-layer gating networks, attention, and normalization. Combined with pre-gated routing for memory-efficient expert prefetching, Megrez2 achieves competitive performance with 7.5B total / 3B active parameters. Megrez2 is a single production system optimised for inference efficiency on memory-constrained devices, and reports one tying configuration without isolating its contribution to model quality. Our work gives a controlled decomposition that varies group size, tying topology, expert granularity, and base architecture (OLMoE, Qwen3-MoE, DeepSeekMoE), and that isolates *which* component of a tied MoE layer is responsible for cross-depth differentiation.

In closely related parallel work, *MoRE* (Qiu et al., 2026) shares an enlarged expert pool across adjacent layers while retaining layer-specific attention and routers, and adds learnable depth embeddings to the inputs of routers and experts. Its parameter-matched pool expansion closely relates to our width-expansion setting. Our complementary focus is a controlled decomposition of attention, router, and expert sharing, together with the trade-off between parameter reduction and reinvestment in expert width.

The looped MoE variant of Chen et al. (2026a) is concurrent work on the special case of fully-looped MoE blocks, sharing all components across iterations and then restoring per-iteration distinctness with an architecture modification, an adaptive-layernorm modulation conditioned on the iteration index. Our component ablation finds that retaining per-layer attention improves loss when experts are shared, beyond retaining distinct layernorm gains alone.

Lee et al. (2026) study fully looped MoE transformers and associate their improved scaling relative to dense baselines with divergent expert selections across loop iterations. Their analysis concerns input-dependent routing decisions under shared parameters, whereas our ablation tests the benefit of untying individual components. In our setting, untying routers substantially changes routing agreement but yields little loss improvement, while untying attention yields substantially larger gains. These findings distinguish routing diversity from the marginal benefit of layer-specific router parameters.

ReXMoE (Tan et al., 2025) approaches cross-layer expert reuse from a complementary angle. Rather than tying expert weights, ReXMoE expands each layer's *candidate expert pool* to include experts

from adjacent layers, with per-layer routers selecting from a union of its own and its neighbors' experts. This increases routing diversity without adding parameters, using a progressive scaling routing (PSR) strategy during training. Our work is complementary: we share the actual weight tensors (reducing unique parameters) while keeping routers per-layer, whereas ReXMoE keeps weights independent while broadening routing scope.

PathMoE (Gu et al., 2026) shares *router* parameters (rather than expert weights) across consecutive layers, arguing that independent per-layer routing over N experts across L layers creates a too large path space of N^L possible expert sequences, while tokens in practice concentrate on a small fraction of coherent linguistic paths. Shared routers constrain the effective path space and improve sample efficiency at 0.9B and 16B scale. PathMoE is the mirror image of our approach—they share routers with distinct experts, while we share experts with distinct routers. A related line of work coordinates routing across layers without sharing any weights: Hu et al. (2026) adds an auxiliary cross-layer coupling loss that aligns top-k routing across adjacent layers, improving expert specialization; this is complementary to weight tying/looping and further supports that cross-layer routing structure is a meaningful axis.

Mixture of Universal Experts (MoUE) (Chen et al., 2026b) takes the sharing idea to its extreme: a single expert pool is shared across *all* layers of the network. To make this work, MoUE introduces a staggered rotational topology for structured expert exposure, a universal load-balancing loss that accounts for repeated expert access across depth, and a universal router with trajectory state to coordinate routing across layers. Where MoUE unifies the expert pool globally, our work studies the intermediate regime of group-wise sharing, which preserves more per-layer flexibility at the cost of less aggressive parameter reduction. Our ablations over group size effectively trace the continuum between per-layer independent experts ($g = 1$) and fully shared experts ($g = L$, MoUE-like).

Global expert-pool designs include *Hyper Experts* (Deriy & Cetin, 2026) and *UniPool* (Huang et al., 2026), both retaining layer-specific routing. UniPool additionally introduces pool-level load balancing and uses NormRouter. More recently, *CS-MoE* (Jiao et al., 2026) combines layer-specific experts with a globally shared expert pool. These approaches explore complementary allocations of expert capacity across depth; our study focuses on group-wise tying within standard MoE architectures and isolates the quality effects of sharing individual components.

WideNet (Xue et al., 2022) is an earlier MoE approach that shares attention, expert, and router parameters across depth while retaining layer-specific normalization. Our study separately varies attention and router sharing to identify their contributions when expert weights are tied.

B Reproducibility details: Component ablations (Section 3)

This appendix documents the full hyperparameter configuration used in the component-tying ablation of Section 3, sufficient to reproduce all reported numbers.

Architecture. We use a vanilla depth-32 decoder-only transformer with $d_{\text{model}} = 512$, $n_{\text{heads}} = 8$, $d_{\text{head}} = 64$, sequence length 512, and a tiktoken `cl100k_base` tokenizer (vocab size 100,277). Each transformer layer is pre-norm (Xiong et al., 2020) with RMSNorm (Zhang & Sennrich, 2019): a multi-head self-attention sub-block—using rotary position embeddings (RoPE) (Su et al., 2024) and QK-norm (Henry et al., 2020) (RMSNorm applied to queries and keys before the RoPE rotation and the dot product)—followed by a sparsely-activated MoE FFN sub-block with SwiGLU activations (Shazeer, 2020). The fine-grained MoE uses 32 experts per layer with top-$k = 8$ and $d_{\text{ff,expert}} = 256$; the coarse-grained variant uses 8 experts with top-$k = 2$ and $d_{\text{ff,expert}} = 1024$, so active FFN width is exactly matched ($8 \times 256 = 2 \times 1024 = 2048$) and total expert parameters per layer are identical. The dense reference uses a single FFN of width 2048, matching the same active FFN compute. Input and output embeddings are tied. All linear projections (attention Q,K,V,O, FFN gate/up/down, router) have biases disabled. Every input sequence starts with a `<BoD>` token, allowing to channel attention-sink behavior.

Tying topologies. The depth-32 stack is partitioned into an untied 2-layer prelude, k tying groups covering the 28 middle layers, and an untied 2-layer coda. Each tying group is a parameter block of b unique transformer layers reused ℓ times consecutively; the group therefore spans $b \cdot \ell$ layers, and the topology overall covers $k \cdot b \cdot \ell = 28$ middle layers. We denote a topology with k groups, block size b, and loop count ℓ as $k\,(\text{group of } b)^{\ell\times}$. The five topologies we sweep are: $1\,(\text{group of } 2)^{14\times}$,

$2\,(\text{group of } 2)^{7\times}$, $3\,(\text{group of } 2)^{(5,4,5)\times}$ (non-uniform loop count 5+4+5=14), $4\,(\text{group of } 1)^{7\times}$, and $7\,(\text{group of } 1)^{4\times}$. The MoE baseline corresponds to no cross-layer tying (all 32 layers carry independent parameters). We additionally evaluate a MoEUT-style topology (Csordás et al., 2024) in which the entire 32-layer stack forms a single tying group ($1\,(\text{group of } 2)^{16\times}$) with no untied prelude or coda; this serves as the maximal-tying extreme of the design space.

Tying modes. Within each group, the chosen subset of components is tied across the group's layers (i.e., each layer in the group references the same parameter tensor for the tied components, while the layers compute forward on distinct activations as in any standard transformer stack). Norm gains are always per-layer in our codebase (each transformer layer has its own RMSNorm scaling parameter) and are therefore not listed as a tying option. We evaluate three modes (Table 1):

- ALL-TIE: FFN expert weights, attention Q,K,V,O projections, and the router are all tied across the group.
- ATTN-TIE: FFN expert weights and attention Q,K,V,O projections are tied; the router is per-layer.
- EXPERT-TIE: only the FFN expert weights are tied; attention and router are per-layer.

Gradient accumulation through tied parameters' multiple use sites is handled natively by PyTorch autograd; no manual gradient manipulation is required.

Optimizer. Following the established recipe of (Jordan et al., 2024; Liu et al., 2025), we use Muon for all 2D hidden weights (attention Q,K,V,O projections, FFN gate/up/down projections) with $\eta_{\text{Muon}} = 2 \times 10^{-2}$, momentum 0.95 with Nesterov, weight decay 0.1, and 5 Newton–Schulz iterations for the orthogonalisation step. The remaining parameters (token and output embeddings, norm gains, router weights, biases) are optimised with AdamW (Loshchilov & Hutter, 2019) at $\eta_{\text{AdamW}} = 2 \times 10^{-3}$ (i.e., $0.1 \times \eta_{\text{Muon}}$, the empirical ratio used by (Jordan et al., 2024)), $\beta_1 = 0.9$, $\beta_2 = 0.95$, and zero weight decay. Routers are kept on AdamW because their output behaves as a per-token classifier head, where adaptive per-parameter learning rates suit the sparse and heavy-tailed gradient distribution.[3]

Tied-parameter LR scaling and weight decay. Every ablation run in this section applies the tied-LR scaling motivated in Section 3.5. In this codebase each expert's gate/up/down projections are stored as separate 2D tensors (stacked into a batched tensor only inside the forward pass), so both the expert weights and the attention Q, K, V, O projections are 2D and are optimised by Muon. Muon parameter groups are bucketed by ℓ, the number of forward use-sites of the shared tensor (the loop count of its tying group), and the learning rate of each bucket is divided by $\sqrt{\ell}$; untied tensors ($\ell = 1$) use divisor 1. In the notation of the topology sweep ℓ is the superscript, so $4\,(\text{group of } 1)^{7\times}$ uses divisor $\sqrt{7} \approx 2.65$ and $7\,(\text{group of } 1)^{4\times}$ uses divisor 2. This is the same rule as the production experiments of Section 4, where a tensor shared by g consecutive layers uses divisor $\sqrt{g}$; only the topology notation differs. Routers, embeddings, the output head and 1D parameters are optimised by AdamW and receive no such scaling; in ALL-TIE, where the router is also shared, the tied router therefore trains without LR compensation. Weight decay is left uncompensated: all Muon groups use the same $\lambda = 0.1$ regardless of ℓ, so the LR division also reduces the per-step shrinkage $\eta\lambda$ for tied parameters. Section 3.5 ablates the divisor choice against the strict $1/\ell$ rule and against no scaling.

Learning rate schedule. Both optimizers follow a cosine schedule with 50-step linear warmup, decaying from peak to $\eta_{\text{min}} = 0.1 \times \eta_{\text{peak}}$ at the final step. The Muon and AdamW peaks decay synchronously.

Loss. Standard next-token cross-entropy plus a load-balancing auxiliary loss (Shazeer et al., 2017; Fedus et al., 2022) with coefficient $\alpha_{\text{aux}} = 10^{-2}$ and a router z-loss with coefficient $\alpha_z = 10^{-4}$. The same loss recipe is used in the production-architecture experiments of Section 4 (Appendix C).

[3] This differs from the convention in some Muon-MoE implementations (Liu et al., 2025) that route 2D router weights through Muon. Our preliminary experiments found neither convention substantially superior at the scales considered here; we use the AdamW-router convention uniformly across all experiments in this paper.

Training. Effective batch size $32 \times 8 \times 512 = 131{,}072$ tokens per optimization step (per-step batch $\times$ gradient accumulation $\times$ sequence length), trained for $10{,}000$ steps (≈ 1.3B tokens total). Gradient clipping at global norm 1.0. Mixed precision: bfloat16 forward and backward with float32 master weights. We enable `torch.compile`.

Data. A 75:25 mixture of DCLM-edu (Allal et al., 2025) and FinePhrase (Niklaus et al., 2026), streamed from HuggingFace with shuffle buffers of $10{,}000$ documents per source and a fixed seed (42) for reproducibility. A held-out 10M-token validation slice (sampled deterministically by skipping the first 10M tokens of the stream before evaluation) is used for all loss reporting.

Random seeds and reruns. Data shuffling uses seed 42; the Section-3 trainer does not explicitly control model-initialization seeds. The additional attention comparisons in Appendix D are independent reruns rather than numbered seed trials. Their within-configuration loss differences are 0.00283 and 0.00409; we do not attribute this variation solely to CUDA nondeterminism. Other sweep cells have one training run each.

Downstream evaluation. We report only validation loss for the ablations of Section 3. The relevant signal in this section is the relative comparison between tying modes at fixed topology; absolute downstream task accuracy at this model scale is below the resolution at which our task suite reliably distinguishes architectural variants. Downstream numbers for our main configurations, evaluated at larger model scale, are reported in Section 4 (Table 4).

Hardware. Each ablation run completes in approximately 3 hours on a single H100 GPU.

C Reproducibility details: Main experiments (Section 4)

This appendix documents the configuration of the production-architecture runs in Table 4, sufficient to reproduce the reported numbers given the released codebase.

Architectures. Scaled-down variants of OLMoE (Muennighoff et al., 2024), Qwen3-MoE (Yang et al., 2025), and DeepSeekMoE (DeepSeek-AI, 2024), instantiated through the HuggingFace `transformers` v5.9 implementations. All configurations in Table 4, including baselines, use the same corrected auxiliary-loss implementation, which consumes raw router logits for the z-loss and softmax probabilities for load balancing. All three are pre-norm (Xiong et al., 2020) with RMSNorm (Zhang & Sennrich, 2019), SwiGLU FFN activations (Shazeer, 2020), and rotary position embeddings (Su et al., 2024); *Qwen3-MoE* additionally applies QK-norm (Henry et al., 2020) (RMSNorm on queries and keys) by default, whereas *OLMoE* and *DeepSeekMoE* use the QKV-clipping mechanism inherited from the OLMoE config (clip threshold 8) instead. Our DeepSeekMoE-style configuration adopts DeepSeek's fine-grained routing (top-6 over 64 experts, without using shared experts). It does *not* use Multi-head Latent Attention (MLA), as the Transformers v5.9 MLA implementation does not support grouped GEMM. Since expert tying operates only on the FFN sub-block and routing, it is independent of the attention variant and is expected to compose well with any alternative attention mechanism, as for example MLA.

OLMoE and *DeepSeekMoE*: $d_{\text{model}} = 512$, 16 layers, 4 attention heads (no GQA), 64 routed experts of FFN intermediate size 256 each, top-$k = 8$ for OLMoE and top-$k = 6$ for DeepSeekMoE; the only difference between the two is top-k, which affects active compute but not total parameter count. *Qwen3-MoE*: $d_{\text{model}} = 384$, 28 layers, 6 attention heads with grouped-query attention (1 KV head), 60 routed experts of MoE intermediate size 192 each, top-$k = 4$. All three architectures use the HuggingFace tiktoken `cl100k_base` tokenizer (vocab size $100{,}277$) with tie_word_embeddings $=$ False. Same as in our smaller ablations setup, every input sequence starts with a `<BoD>` token, allowing to channel attention-sink behavior.

Active and total parameter counts per configuration are reported in Table 4.

Tying topology and width expansion. For each architecture the five configurations are: *baseline* (g=1, no tying); g=2 and g=4 (expert FFN tensors aliased across consecutive groups of 2 or 4 middle layers, 2-layer prelude and 2-layer coda untied); g=4, 2× *width* and g=4, 4× *width*; the latter two double or quadruple the number of experts in the tied middle layers, leaving prelude and coda at the baseline expert count. The $4\times$ variant returns each architecture's total parameter count to within 1% of its baseline ("iso-base"). Heterogeneous width is implemented by constructing a

temporary model with the expanded expert count and transplanting its middle-layer MLPs into the main model before applying expert tying.

Optimizer. The same Muon/AdamW configuration as the ablation runs (Appendix B), except for the AdamW weight decay and the warmup length, both noted below: Muon (Jordan et al., 2024; Liu et al., 2025) for 2D hidden weights at $\eta_{\text{Muon}} = 2 \times 10^{-2}$ with weight decay 0.1, momentum 0.95 Nesterov, 5 Newton–Schulz iterations; AdamW (Loshchilov & Hutter, 2019) optimizes the embeddings, output head, norm gains, biases, and routers at $\eta_{\text{AdamW}} = 0.1 \cdot \eta_{\text{Muon}} = 2 \times 10^{-3}$ with $(\beta_1, \beta_2) = (0.9, 0.95)$. We apply a uniform weight decay of 0.01 to all AdamW parameters, whereas the ablation runs of Appendix B use 0; each setting is held fixed across all configurations within its own set of experiments, so it does not affect any comparison made within a table. While some large-scale Muon-MoE recipes apply heavy regularization (0.1) to AdamW parameters (Team et al., 2025), we adopt the lighter 0.01 standard from the foundational Muon baseline (Jordan et al., 2024). For the routers specifically, this provides a gentle physical shrinkage that works in tandem with the auxiliary z-loss to prevent logit explosion and softmax saturation.

The difference between our nominal $\eta_{\text{Muon}} = 2 \times 10^{-2}$ and the $\eta \approx 2 \times 10^{-4}$ reported by Liu et al. (2025) and Kimi K2 (Team et al., 2025) must be interpreted in light of their different update-scaling conventions. Liu et al. (2025) multiply the orthogonalised update by $0.2 \cdot \sqrt{\max(d_{\text{in}}, d_{\text{out}})}$, which cancels the $1/\sqrt{\max(d_{\text{in}}, d_{\text{out}})}$ per-entry RMS of the orthogonalised update and leaves a shape-independent per-entry update RMS of $0.2\,\eta$. The original Jordan recipe (Jordan et al., 2024) we use applies $\sqrt{\max(1, d_{\text{out}}/d_{\text{in}})} \approx 1$ for the near-square FFN matrices in our architectures, leaving a per-entry update RMS of $\eta_{\text{Muon}}/\sqrt{\max(d_{\text{in}}, d_{\text{out}})}$. For our largest ablation matrices ($\max d \approx 1024$), this gives $\sim 6 \times 10^{-4}$ per entry, against 4×10^{-5} per entry for K2. Comparing update magnitudes therefore requires accounting for both the matrix dimensions and the update-scaling convention.

3D expert tensors and tied-LR scaling. The HuggingFace MoE implementations stack expert weights into 3D tensors of shape $(E, 2d_{\text{ff,expert}}, d_{\text{model}})$ (fused gate/up) and $(E, d_{\text{model}}, d_{\text{ff,expert}})$ (down). Muon expects 2D inputs, so we register one 2D proxy parameter per expert that shares storage with its slice of the 3D tensor; gradients are copied from the 3D parameter into the proxies via a post-backward hook before the optimiser step. For *tied* expert tensors, autograd sums the gradients from all g layer use-sites into the shared parameter. We divide the learning rate of the tied-expert Muon parameter groups by $\sqrt{g}$ (`tied_lr_divisor=2.0` for g=4), the rule selected empirically in Section 3.5. Only expert tensors are tied in these experiments, and all of them are optimised through the Muon proxies, so no AdamW parameter group is affected; routers, embeddings, and norm gains are untied and use divisor 1, as does the untied baseline. In preliminary runs, omitting the scaling degraded early-step stability and final loss.

Decoupled optimizers apply weight decay as a penalty proportional to the learning rate: $W \leftarrow W - \eta\, u - \eta\lambda W$, where u is the optimiser's update direction. Dividing η by $\sqrt{g}$ for tied parameters therefore also reduces their per-step regularization $\eta\lambda$ by the same factor. We leave this uncompensated: all parameter groups, tied and untied, use the same base λ with no $\sqrt{g}$ adjustment, so tied parameters receive proportionally less structural shrinkage per step. In an ablation at g=4 we found that compensating the decay (multiplying λ by $\sqrt{g}$ for tied groups, restoring the untied $\eta\lambda$) gave *worse* final loss than leaving it uncompensated, so the simpler uniform-λ scheme is used throughout.

Learning rate schedule. Cosine schedule with 100-step linear warmup, decaying from η_{peak} to $\eta_{\text{min}} = 0.1 \times \eta_{\text{peak}}$ at the final step; Muon and AdamW peaks decay synchronously (and the tied-LR-divided groups inherit the same shape).

Loss. Standard next-token cross-entropy plus a load-balancing auxiliary loss (Shazeer et al., 2017; Fedus et al., 2022) with coefficient $\alpha_{\text{aux}} = 10^{-2}$ and a router z-loss with coefficient $\alpha_z = 10^{-4}$. Identical to the ablation runs of Section 3 (Appendix B).

Training and data. Training used $\approx$10.5B tokens total for the smaller configs, and $\approx$15.7B tokens for the 7B configs. This corresponds to 20,000 or 30,000 optimisation steps respectively. All configs use a global batch size of 256 sequences per step, or 524,288 tokens, at 2048 sequence length. All small configurations use microbatch size 16 and, with four DDP ranks, gradient accumulation 4. The full-scale baseline, g=4, and g=4 with 2$\times$ width use per-rank microbatches 4, 16, and

8, with accumulation 16, 4, and 8, respectively, holding the global batch fixed. For additional single-GPU throughput experiments, the same micro-batch size results in grad accumulation 16. Gradient clipping is active at a global norm 1.0. Mixed precision: bfloat16 forward and backward, float32 master weights, `torch.compile` enabled where available. Larger configs use gradient checkpointing for saving GPU memory. We use the same 75:25 DCLM-edu (Allal et al., 2025) and FinePhrase (Niklaus et al., 2026) data mixture as the ablations (Appendix B), with the same data loader, tokenised with the same `cl100k_base` encoding.

Hardware. Each run uses 4×H200 GPUs with PyTorch DDP. For the smaller config in Table 4, each 20,000-step wall-clock time per run is roughly 5 hours. The larger 7B scale experiments with 30,000-step as in Table 5 takes about 3.5 days (g=4 tied setting) on the 4 GPUs.

Efficiency & Throughput. We measure sustained training throughput at the 1,000-step mark on 4×H200 GPUs with PyTorch DDP. Full-scale untied OLMoE processes 41,859 tokens/sec; g=4 processes 51,777 tokens/sec (+23.7%). The tied model fits a fourfold larger microbatch at the same global batch, so this is an end-to-end operating-point comparison, not a microbatch-matched kernel benchmark. Reduced-scale OLMoE uses matched microbatches and processes 200,579 versus 232,171 tokens/sec (+15.7%). Communication uses intra-node NVLink/NVSwitch. These measurements do not isolate the individual contributions of optimizer work, gradient communication, and batching.

Under DDP, tying reduces the number of unique gradient elements to all-reduce. For expert or tensor parallelism, storage savings require reusing expert placement or shards across tied depths; token dispatch and tensor-parallel collectives remain per-layer. Pipeline-parallel tying groups can stay within a stage; splitting a group across stages requires replication or remote access. These are implementation considerations, not measured EP/TP/PP throughput results. We have not measured training peak memory or inference latency. Kernels specialized for tied layers are not used; their potential effect on throughput is untested.

Downstream evaluation. We report macro-average 3-shot accuracy on {ARC-Easy, ARC-Challenge, HellaSwag, PIQA, WinoGrande, OpenBookQA} via `lm-evaluation-harness`, using plain `acc` for every task on the final saved checkpoint of each run. Main comparisons have one training run per configuration. Appendix F reports a separate extended evaluation and its per-task sampling uncertainty; that uncertainty does not measure pretraining-run variance.

D Additional component controls

D.1 Independent attention reruns

Table 6 reports two independent runs of each configuration in the 2-group, fine-grained topology underlying the conditional attention comparison in Table 2. The trainer does not explicitly control initialization seeds, so these are reruns rather than trials at specified seeds. In both comparisons, additionally tying attention increases loss by substantially more than the observed within-configuration variation. This does not establish the effect of tying attention when experts remain untied.

Table 6: Independent reruns of the attention comparison at 2-group, fine-grained. Expert FFNs are shared and routers are per-layer in both configurations. Loss is validation next-token cross-entropy.

Configuration	Run A	Run B
Experts tied; attention per-layer	3.46895	3.46612
Experts and attention tied	3.50865	3.51274
Additional attention-tying penalty	+0.03970	+0.04662

Cross-loop agreement is 0.09993 and 0.12016 with per-layer attention, versus 0.17652 and 0.18967 with attention also tied. Within-configuration loss differences are 0.00283 and 0.00409; two runs per configuration do not provide a precise estimate of training variance across the wider sweep.

D.2 Matched boundary placement

The five configurations in Table 7 have identical unique parameters (186.1M), effective depth (32), active computation, repeat-count multiset $\{1, 1, 1, 1, 7, 7, 7, 7\}$, and tied-LR treatment. Only the

depth ordering changes. A 1 denotes an untied singleton and a 7 denotes a shared expert block used seven times; attention and routers remain layer-specific.

Table 7: Matched placement of four untied layers. Each row has one training run; smaller pairwise differences are comparable to the variation observed in the independent reruns.

Untied-layer placement	Depth-order repeat profile	Loss
One per end + two interior	$[1, 7, 1, 7, 7, 1, 7, 1]$	3.4667
Coda only + two interior	$[7, 7, 1, 7, 1, 7, 1, 1]$	3.4683
2+2 boundaries	$[1, 1, 7, 7, 7, 7, 1, 1]$	3.4689
Prelude only + two interior	$[1, 1, 7, 1, 7, 1, 7, 7]$	3.4712
Interior only	$[7, 1, 7, 1, 1, 7, 1, 7]$	3.4734

The all-interior placement has the highest observed loss, but the total spread is below 0.007. The experiment does not resolve a reliable ordering among boundary variants, establish that $2{+}2$ is necessary or optimal, or isolate an asymmetric prelude/coda mechanism. The one-per-end variant retains two additional untied interior layers, so it is a placement control rather than a reduction from four untied layers to two.

D.3 Dense FFN-sharing control

We train two dense models in the depth-32 ablation setup with $d_{\text{ff}} = 2048$ for $10{,}000$ steps under the same data and optimization recipe. The tied model shares FFNs within four recurrent middle groups, retaining per-layer attention and the $2{+}2$ untied boundaries. Effective depth, FFN width, and per-token FFN computation are fixed within this dense comparison.

Table 8: Dense FFN-sharing control. The loss increase is relative to the untied dense baseline.

Dense configuration	Unique params	Loss	Δ loss
Untied	185.6M	3.5597	—
FFNs shared; attention per-layer	110.1M	3.6291	+0.0694

Dense FFN sharing removes 75.5M parameters (40.7%) at a loss cost of 0.0694. For context, the fine-grained four-group EXPERT-TIE row in Table 2 removes 61.9% of its MoE baseline's parameters at a loss cost of 0.038. The dense width equals the MoE's selected width ($8 \times 256 = 2048$), approximately matching dominant FFN arithmetic, while the MoE incurs additional routing and dispatch. Because total capacity and architecture differ, this comparison does not isolate a causal benefit of routing.

E Measured inference memory

Table 9 reports full-scale OLMoE measurements on one H200 in float32. Resident-weight storage follows the unique parameter reduction; peak memory during evaluation also includes non-weight allocations and therefore falls by a smaller fraction. For $g{=}4$, peak reductions are 41.6%, 26.2%, and 42.4% in the three reported phases. These are measurements under this evaluation setup, not inference-latency or training-memory results.

Table 9: Full-scale inference memory in GiB (one H200, float32). The three peak columns correspond to the six-task 3-shot phase, MMLU continuation, and TriviaQA evaluation.

Configuration	Weights	Peak: 3-shot	Peak: MMLU	Peak: TriviaQA
Untied	26.54	32.48	51.56	31.83
$g{=}4$	13.04	18.98	38.06	18.33
$g{=}4$, $2\times$ width	17.54	23.49	42.57	22.84

F Extended downstream evaluation and uncertainty

Table 10 reports a separate extended evaluation of the final full-scale OLMoE checkpoints. The seven-task accuracy average uses `acc_norm` where available and adds MMLU continuation; it is not directly comparable to the six-task plain-`acc` averages in the main tables. MMLU continuation scores answer-text likelihood rather than answer-letter likelihood and is not comparable to standard published MMLU scores. TriviaQA adds generative closed-book factual recall, reported as exact match (EM) separately from the accuracy average.

Table 10: Extended evaluation of full-scale OLMoE checkpoints after 15.7B training tokens. Values are percentages; $\pm$ denotes one evaluation-set standard error in percentage points, not training-run uncertainty. The macro-average excludes TriviaQA.

Task	Untied	g=4	g=4, 2× width
ARC-Challenge	33.70 ± 1.38	34.04 ± 1.38	34.47 ± 1.39
ARC-Easy	65.24 ± 0.98	63.97 ± 0.99	65.49 ± 0.98
HellaSwag	49.35 ± 0.50	49.18 ± 0.50	49.94 ± 0.50
PIQA	70.84 ± 1.06	71.55 ± 1.05	69.48 ± 1.07
WinoGrande	54.30 ± 1.40	55.41 ± 1.40	57.22 ± 1.39
OpenBookQA	35.60 ± 2.14	35.60 ± 2.14	34.00 ± 2.12
MMLU continuation	30.99 ± 0.38	30.83 ± 0.38	31.35 ± 0.39
Accuracy macro-average	48.57	48.65	48.85
TriviaQA closed-book EM	15.85 ± 0.82	16.25 ± 0.83	16.85 ± 0.84

Per-task changes are mixed, and small gaps should not be interpreted as demonstrated improvements or formal equivalence. Each checkpoint comes from one pretraining run. Repeating the g=4 evaluation reproduced every score, establishing fixed-checkpoint reproducibility only; it does not quantify pretraining-run variance.

G Monitoring Router Health and Expert Utilization

A common failure mode in MoE training is router collapse, where the gating network degenerates to selecting a small subset of experts and the model effectively reduces to a dense network (Shazeer et al., 2017). Because our architecture forces consecutive layers to share the same underlying expert weights, it is important to verify that tying does not encourage collapse or degrade routing diversity.

Logged metrics. For every run in Sections 3 and 4 we track, averaged across all routing layers:

- **Auxiliary load-balancing loss** (Shazeer et al., 2017; Fedus et al., 2022) ($\alpha_{\text{aux}} = 10^{-2}$), which penalises imbalanced expert usage.
- **Mean routing entropy:** the Shannon entropy of the softmax-normalised routing distributions; low entropy indicates the router is committing to few experts.
- **Router z-loss** (Zoph et al., 2022) ($\alpha_z = 10^{-4}$), which tracks the magnitude of the pre-softmax router logits and guards against logit explosion.
- **Cross-loop agreement:** for tied configurations, the per-token top-1 routing agreement between layers that share an expert (the same metric defined in Section 3). High agreement indicates the shared expert is being reused with near-identical routing across loop positions; lower agreement indicates genuinely different per-layer use.

Note that the router parameters receive weak weight decay in the AdamW split (0.01 in Section 4; none in the Section 3 ablations shown in Figure 1; see Appendices B and C), so they undergo little or no structural shrinkage; the z-loss guards against logit explosion in its absence.

Observations. Across all tying topologies the router remains healthy throughout training: the auxiliary loss stays bounded (it does not run away or collapse to zero), routing entropy stays well above zero, and the z-loss keeps router logits in a stable range rather than diverging. Tying does not induce router collapse at any group size.

Full-scale diagnostics. In the OLMoE-1B-7B experiment, the fraction of experts that are never any token's top-1 choice in a logged batch is about 8% for both $g=4$ and the untied baseline (7.7% and 8.6%). Cross-loop agreement — the normalised overlap of top-8 expert sets across the four layers of a tied group — stabilises around 0.51. Each token is therefore routed to about 20 distinct experts across the tied layers, against 8 if routing were identical and about 26 under independent random choice. Routing is correlated across depth but far from identical, even though the expert weights are shared.

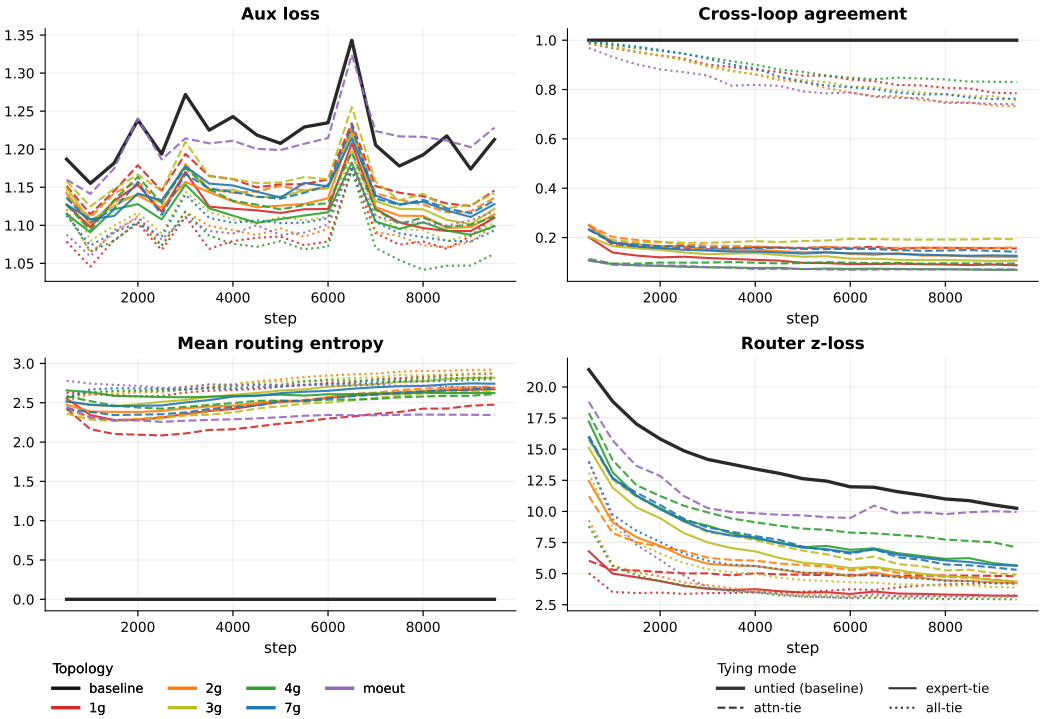

Figure 1: Router-health metrics across the Section 3 fine-grained ablation runs, under the $\sqrt{g}$ tied-LR scaling. **Colour** encodes topology (baseline = black, $1g$ = red, $2g$ = orange, $3g$ = olive, $4g$ = green, $7g$ = blue, MoEUT = purple); **line style** encodes tying mode (ATTN-TIE dashed, EXPERT-TIE solid, ALL-TIE dotted, untied baseline thick solid). The auxiliary loss, routing entropy, and z-loss remain stable across all configurations, indicating no router collapse under tying. EXPERT-TIE tracks ATTN-TIE closely on cross-loop agreement and entropy—shared experts are reused with genuinely different routing across loop positions—while ALL-TIE shows elevated cross-loop agreement, as expected when the router itself is also shared.