AHAMOE: EXPERT ROUTING AHEAD OF LANGUAGE MODEL COMPUTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Routing from intermediate hidden states limits expert prefetching in Mixture-of-Experts (MoE) models. Existing methods that route from input tokens freeze their routing policies, although the model continues to learn. Once a standalone router takes over expert selection, where can it obtain supervision to continue learning? We show that gates trained only to weight expert outputs can still provide this supervision. We introduce *AhaMoE*, which combines router initialization and partial parameter reuse from an early source model, and continual adaptation of the routing policy to determine expert assignments for all layers before model computation. Experiments with 2.6B-parameter target models trained on 32B tokens show that AhaMoE outperforms existing pre-routed baselines and matches or outperforms hidden-routed baselines without partial parameter reuse in language modeling and zero-shot evaluation. Controlled experiments further show that our partial-reuse strategy improves hidden-routed models as well.

1 INTRODUCTION

Mixture-of-Experts (MoE) has become a standard architectural component of leading large language models (LLMs), including DeepSeek-V4, Kimi K3 and GLM 5 (DeepSeek-AI et al., 2026; Team et al., 2026a; GLM-5-Team et al., 2026), for being able to increase model capacity without increasing model computation. However, it comes with excessive memory consumption to store all expert weights. As such, expert offloading is typically applied to reduce GPU memory requirements by storing some expert weights in CPU memory or on disk (Chen et al., 2025a; Kamahori et al., 2025; Cao et al., 2025; Hwang et al., 2025; Fang et al., 2025; Kim et al., 2026) and overlapping their transfer with computation. However, prefetching is difficult in conventional MoE models because expert selection depends on the token's hidden state at each layer, which is unavailable until the preceding computation is complete. Aside from predicting expert selections (Xue et al., 2025; Fang et al., 2026; Zhu et al., 2025; Tang et al., 2024; Chen et al., 2025b; Wang et al., 2025a) which can be incorrect and cause bandwidth waste, another branch is to *decide*, rather than predicts, the experts that the language model will execute. We call this *pre-routed MoE model*, which uses a standalone router to determine expert assignments from input tokens alone, before model computation begins. Then, an offloading runtime can schedule transfers without waiting for any layer's hidden state.

Existing methods, including Hash Layers (Roller et al., 2021), StableMoE (Dai et al., 2022), and Grouter (Xu et al., 2026), already enable routing from input tokens, but incur a loss in model quality relative to modern hidden-routed MoE models. Specifically, Hash Layers uses a fixed token-to-expert mapping, while StableMoE and Grouter learn standalone student routers by distilling the routing decisions of a hidden-routed MoE model. In particular, StableMoE has two training stages. In the first stage, a hidden-routed MoE model is trained while a student router distills its routing decisions at each training step. In the second stage, the student router is frozen and takes over expert selection, turning the model into a pre-routed MoE. Grouter distills a student router from an existing hidden-routed MoE model and then uses it to train a new pre-routed MoE model, keeping the student router frozen throughout this training.

In all three methods, the routing policy remains fixed during the pre-routed training stage. Modern MoE models, by contrast, continually update their routers to adapt to changes in the rest of the model (DeepSeek-AI et al., 2026; GLM-5-Team et al., 2026; Team et al., 2026a). Hash Layers provides no learned mechanism for adapting its fixed token-to-expert mapping. For a distilled student

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.

router, continued adaptation during the pre-routed training stage raises a different question. It is natural to distill a student router from gates that themselves perform routing, as in StableMoE and Grouter. However, during the pre-routed training stage, these gates no longer act as routers: expert selection is controlled by the student itself. Once the student takes over routing, where can it obtain supervision to continue learning?

Our key observation is that a conventional hidden-state gate receives no gradients through the discrete expert-selection operation itself. Instead, the gate learns to select experts by learning how to weight the outputs of the selected experts. This suggests that gates trained only to weight expert outputs may still teach a student router how to select experts. We use these gate logits to supervise continual router adaptation. Our controlled experiments show that adapting the routing policy yields better final model quality than keeping it fixed.

Training such a router, however, introduces a *bootstrapping* problem. The student router learns its routing policy from the target model's gates, while the target MoE model is trained under the expert assignments produced by the student. Learning a good student router therefore depends on useful supervision from the gates, while training the target model depends on a good student router.

To address this bootstrapping problem, we initialize the student router by distilling a hidden-routed source MoE model trained on only a small fraction of the target model's full token budget. This gives the student a learned routing policy before target-model training begins. The target model can then adapt to this policy from the outset, accelerating the bootstrapping process. We also develop a partial-reuse strategy that retains the embedding and shallow layers of an early source MoE while reinitializing the remaining layers. Our controlled experiments show that this strategy improves model quality under both hidden routing and pre-routing. Partial parameter reuse and router initialization share the same source model, allowing us to combine their benefits while incurring the source-training cost only once.

In this paper, we combine these strategies into *AhaMoE*, a training recipe for pre-routed MoE models. This recipe enables pre-routed models to match or outperform hidden-routed baselines that do not use our partial-reuse strategy. In *AhaMoE*, a standalone router selects experts for all layers using only the input token sequence, simplifying the design of inference systems with expert offloading. We first train a hidden-routed source MoE model using only a small fraction of the target model's full token budget. We then initialize part of the target model with its parameters and initialize the standalone router by distilling the logits produced by its gates. During target-model training, the standalone router continues to learn at each step from the logits produced by the target model's MoE gates, which no longer perform routing.

The rest of the paper is organized as follows. Section 2 reviews modern MoE architectures and baselines closely related to our method. Section 3 presents *AhaMoE*, including the architecture and initialization of the standalone router and target model, and the target-model training procedure. Finally, Section 4 evaluates our method's performance and efficiency through extensive experiments.

2 Background

2.1 Fine-grained MoE and loss-free balancing

An MoE model replaces dense feed-forward layers with MoE layers. Each MoE layer contains multiple feed-forward networks, called experts, and a router that selects a small subset of experts for each token. Fine-grained MoE uses smaller experts, allowing each token to activate more experts under the same computation budget. In architectures with shared experts, these experts process every token regardless of the router's selections (Dai et al., 2024). We adopt a hidden-routed MoE architecture with sigmoid affinities, normalized top-k mixture weights, shared experts, and loss-free load balancing (DeepSeek-AI et al., 2025a).

Fine-grained MoE with shared experts. We describe a single token at a single MoE layer and omit the token and layer indices for simplicity. Let $\boldsymbol{h} \in \mathbb{R}^d$ be the hidden state supplied to the experts after pre-MoE normalization. Each layer has N_r routed experts and N_s always-active shared experts. We denote the j-th routed expert by $E_j^{\text{routed}} : \mathbb{R}^d \to \mathbb{R}^d$ and the i-th shared expert by $E_i^{\text{shared}} : \mathbb{R}^d \to \mathbb{R}^d$. Let $\boldsymbol{u}_j \in \mathbb{R}^d$ and $b_j \in \mathbb{R}$ be the learned gate vector and balancing bias of

expert j, respectively, and let σ denote the sigmoid function. For k selected routed experts, the MoE branch output $\boldsymbol{y}$ before residual addition is computed as

$$s_j = \sigma(\boldsymbol{u}_j^\mathsf{T}\boldsymbol{h}), \qquad \mathcal{S} = \mathrm{TopK}_{j\in\{1,\dots,N_r\}}(s_j + b_j, k), \qquad p_j = \frac{s_j}{\sum_{j\in\mathcal{S}} s_j},$$
$$\boldsymbol{y} = \sum_{i=1}^{N_s} E_i^{\mathrm{shared}}(\boldsymbol{h}) + \sum_{j\in\mathcal{S}} p_j E_j^{\mathrm{routed}}(\boldsymbol{h}). \tag{1}$$

Loss-free balancing. Imbalanced expert assignments can concentrate work on a few experts. Loss-free balancing controls these assignments by adjusting b_j directly, without adding a balancing term to the language-model loss (DeepSeek-AI et al., 2025a). Let n_j be the number of tokens assigned to routed expert j, aggregated across workers and accumulated over one optimizer step, and let $\bar{n} = \sum_{j=1}^{N_r} n_j/N_r$ be the mean expert load. With bias update rate γ, b_j is updated as follows:

$$b_j \leftarrow b_j + \gamma\ \mathrm{sign}(\bar{n} - n_j). \tag{2}$$

This raises the biases of underloaded experts and lowers those of overloaded experts. The biases start at zero, are updated outside gradient descent once per optimizer step during training, and remain fixed at inference.

The prefetching window. For expert offloading, prediction-based prefetching can begin before $\mathcal{S}$ is known but may fetch the wrong experts. Prefetching without prediction must wait until $\mathcal{S}$ is determined. In Equation equation 1, $\mathcal{S}$ depends on $\boldsymbol{h}$ through s_j, leaving only affinity normalization and any shared-expert computation scheduled before the routed experts to overlap with weight transfers. If $\mathcal{S}$ depends only on input tokens, expert selection for all layers can precede model computation, allowing transfers to overlap with all computation preceding each layer.

2.2 Routing from input tokens

Several existing methods already determine expert assignments from input tokens without waiting for intermediate hidden states. We review three such methods: Hash Layers, StableMoE, and Grouter.

Hash Layers. Hash Layers replaces learned expert selection with a fixed mapping from input token identities to experts (Roller et al., 2021). In its basic form, each token id is assigned to one expert before training, and every occurrence of that type uses the same expert. The expert still processes the current hidden state, but its identity can be looked up directly from the input token. However, its routing decisions cannot depend on preceding tokens or adapt as the model trains. These limitations are less consequential in shallow layers, where balancing learned routing can be difficult and fixed assignments can help stabilize training (Su, 2026). Thus, DeepSeek-V4 uses hash-based routing in its first three MoE layers (DeepSeek-AI et al., 2026). In our experiments, using Hash Layers in the first few layers also yields slightly lower training loss.

We adapt Hash Layers to fine-grained top-k MoE by assigning k distinct expert IDs to each token ID at each layer, taking load balance into account. Appendix C describes the assignment procedure. All Hash Layers used in this paper follow this adaptation.

StableMoE. StableMoE learns a standalone router through a two-stage training procedure (Dai et al., 2022). During the first stage, a hidden-state gate selects one expert per token and trains jointly with the language model. In parallel, a student router uses input token embeddings to imitate these expert indices with a cross-entropy distillation loss. At the stage boundary, the student takes over expert selection and is frozen for the remaining second stage training and inference. The original hidden-state gate continues to supply the sigmoid weight applied to the selected expert's output. Thus, StableMoE fixes the token-to-expert assignments while retaining trainable hidden-state output weighting.

StableMoE compares several student router architectures and finds that token embeddings perform best. However, this comparison uses a single MoE layer inserted in the middle of the model, with each expert containing multiple stacked FFNs selected by a single routing decision (Dai et al., 2022).

This architecture differs substantially from modern fine-grained MoE, which routes separately at successive layers. The preference for an embedding-only router therefore need not carry over to this setting, and such a router cannot represent information from preceding tokens. Continued distillation after student takeover poses another limitation: the original loss uses the actually selected expert ID as its label. After takeover, that ID is produced by the student itself, so the loss no longer provides an independent teacher routing target.

We adapt StableMoE's router architecture, balancing method, and distillation loss to our fine-grained top-k MoE baseline, as detailed in Appendix D.

Grouter. Grouter obtains its standalone router before target-model training by distilling a separately trained source MoE (Xu et al., 2026). Its router comprises token embeddings, Transformer encoder blocks, and an expert-score projection. It learns from the source model's first MoE layer by minimizing the KL divergence between the softmax distributions of source and student logits. After calibrating the final projection for load balance on target data, Grouter freezes the router during target-model training. A single shared router supplies expert indices and mixture weights to all target MoE layers, making these quantities available directly from the input sequence.

This design has several limitations. Supervising only the first layer poses another limitation. Our layer-wise results in Appendix G.2 show that linear gates applied directly to frozen source embeddings predict the first layer's routing accurately, but prediction error increases sharply from the second layer. Distilling only the first layer therefore omits supervision from deeper routing policies that are harder to recover from token embeddings alone. Finally, because the standalone router also weights expert outputs, the target model no longer retains hidden-state MoE gates that could supervise continued learning of the standalone router.

We adapt Grouter's router architecture, distillation loss, and calibration to our fine-grained top-k MoE baseline, as detailed in Appendix E.

These limitations motivate a standalone router that captures causal context, supports distinct routing policies across layers, and introduces little computational and memory overhead. Beyond its architecture, the router needs a training procedure that allows its routing policy to adapt as the target model learns.

3 METHODS

In this section, we present *AhaMoE*, covering the standalone router architecture, the target model's architecture and initialization, the initialization of the standalone router, and the training procedure of the target model.

3.1 ROUTER ARCHITECTURE

AhaMoE must determine expert assignments for an entire MoE model from its input tokens while adding little computation and memory overhead. This requires a router that captures contextual information and learns how routing decisions depend on that context at different model depths. We conduct extensive controlled experiments to choose the router architecture used in AhaMoE.

Each router layer consists of a Kimi Delta Attention (KDA) (Team et al., 2025) sublayer and a SwiGLU feed-forward network (FFN). For an input token x, let $\mathrm{Emb}(x) \in \mathbb{R}^d$ denote its embedding in the MoE model, and let d_r denote the router's hidden size. We apply stop-gradient, denoted by sg, and a trainable projection $\boldsymbol{P} \in \mathbb{R}^{d_r \times d}$ to obtain the router input embedding:

$$\boldsymbol{e} = \boldsymbol{P}\,\mathrm{sg}(\mathrm{Emb}(x))\,. \tag{3}$$

At inference, we can precompute $\widetilde{\mathrm{Emb}}(x) = \boldsymbol{P}\,\mathrm{Emb}(x)$ for every token ID and store the results as a router embedding table, eliminating the separate projection parameters and computation.

For a model with L MoE layers and a router with L_r layers, let ℓ and m index their respective layers. Let $\boldsymbol{r}^m \in \mathbb{R}^{d_r}$ denote the router's hidden state for the current token after router layer m. Assuming that L is divisible by L_r, we use grouped readouts, dividing the MoE layers into L_r consecutive groups of size $G = L/L_r$. MoE layer ℓ reads from router layer $g(\ell) = \lceil \ell/G \rceil$. For each MoE layer

ℓ, the j-th routed expert has a learned vector $\boldsymbol{v}_j^\ell \in \mathbb{R}^{d_r}$, which produces its router logit

$$z_j^\ell = \boldsymbol{v}_j^{\ell\,\mathsf{T}} \boldsymbol{r}^{g(\ell)}. \tag{4}$$

Appendix G provides the remaining architectural details and controlled comparisons.

3.2 Training a source MoE

Let $\mathcal{M}_\mathrm{s}$, $\mathcal{R}$, and $\mathcal{M}_\mathrm{t}$ denote the source MoE, standalone router, and target MoE, with parameters $\boldsymbol{\theta}_\mathrm{s}$, $\boldsymbol{\theta}_\mathrm{r}$, and $\boldsymbol{\theta}_\mathrm{t}$, respectively. The source and target models share the fine-grained top-k MoE architecture described in Section 2.1, with L MoE layers and the same expert configuration. Both use hash layers in their L_h layers. In the remaining layers, the source model uses hidden-state gates to select experts, whereas the target model uses the standalone router $\mathcal{R}$. $\mathcal{M}_\mathrm{t}$ retains its hidden-state gates to weight expert outputs and provide supervision for $\mathcal{R}$.

Let B_s and B_t denote the training budgets of $\mathcal{M}_\mathrm{s}$ and $\mathcal{M}_\mathrm{t}$ in tokens, respectively. We train the source model on a small fraction of the budget for $\mathcal{M}_\mathrm{t}$, with $B_s \ll B_t$, using the language-model objective. Its training data are drawn from the same data used to train $\mathcal{M}_\mathrm{t}$. The resulting source model provides both supervision for router initialization and parameters for target initialization.

3.3 Initializing the standalone router

We freeze the source model $\mathcal{M}_\mathrm{s}$ and train the standalone router $\mathcal{R}$ to match the logits produced by the hidden-state gates in its non-hash layers. We describe the loss for one token at one layer, omitting token and layer indices. Let a_j be the gate logit produced by $\mathcal{M}_\mathrm{s}$ for routed expert j, and let z_j be the corresponding logit produced by $\mathcal{R}$. The sigmoid affinities are $s_j = \sigma(a_j)$ and $\hat{s}_j = \sigma(z_j)$. We treat each affinity produced by $\mathcal{M}_\mathrm{s}$ as a soft binary label and minimize binary cross-entropy (BCE):

$$\mathcal{L}_\mathrm{BCE} = -\frac{1}{N_r} \sum_{j=1}^{N_r} \left[s_j \log \hat{s}_j + (1 - s_j) \log(1 - \hat{s}_j) \right]. \tag{5}$$

This loss differs from Bernoulli KL divergence only by a constant and yields the same gradients with respect to $\boldsymbol{\theta}_\mathrm{r}$. The affinities produced by $\mathcal{M}_\mathrm{s}$ are detached, and the loss is averaged over valid tokens and non-hash layers. After distillation, we freeze $\boldsymbol{\theta}_\mathrm{r}$ and calibrate per-expert selection biases on training data using quantile balancing (Team et al., 2026a). This calibration improves the load balance of the initialized router without changing $\boldsymbol{\theta}_\mathrm{r}$. Appendix F describes the calibration algorithm.

3.4 Initializing the target MoE

We initialize the target model by copying the embedding and first L_h complete Transformer layers of $\mathcal{M}_\mathrm{s}$, together with their hash tables. The remaining layers and output head are initialized afresh, and all of $\boldsymbol{\theta}_\mathrm{t}$, including the parameters of the inherited embedding and layers, are updated from the first training step. Optimizer states and training schedules are also initialized afresh.

By default, we reuse exactly the first L_h hash layers. All MoE layers controlled by the learned router therefore start with fresh parameters, including both gates and experts, and can adapt to the distilled initial routing policy. This avoids forcing pretrained experts to adapt abruptly to a different routing policy. To see why, consider reusing the entire source model: its pretrained experts would switch from routing by the gates of $\mathcal{M}_\mathrm{s}$ to routing by $\mathcal{R}$, undergoing the same kind of routing transition that occurs when StableMoE's student takes over. We therefore conservatively reuse only the hash layers to avoid this routing transition.

3.5 Training the target model

The initialized standalone router controls expert selection from the first training step of $\mathcal{M}_\mathrm{t}$ and continues learning from the target model's gates throughout training. For a non-hash layer, let z_j and b_j denote the standalone router's logit and selection bias for routed expert j, respectively. The selected expert set is:

$$\mathcal{S} = \mathrm{TopK}_{j \in \{1,\ldots,N_r\}} \big(\sigma(z_j) + b_j, k \big). \tag{6}$$

Algorithm 1 Training the target model in AhaMoE

Require: Initialized $\mathcal{M}_{\mathrm{t}}$, $\mathcal{R}$, and selection biases $\boldsymbol{b}$

for each training batch $\mathcal{B}$ **do**
  $\boldsymbol{g}_{\mathrm{t}}, \boldsymbol{g}_{\mathrm{r}} \leftarrow \mathbf{0}, \mathbf{0}$
  $\mathcal{S}, \boldsymbol{z}, \boldsymbol{n} \leftarrow \mathcal{R}(\mathcal{B})$
  $\mathcal{L}_{\mathrm{LM}}, \boldsymbol{a} \leftarrow \mathcal{M}_{\mathrm{t}}(\mathcal{B}, \mathcal{S})$
  $\mathcal{L}_{\mathrm{distill}} \leftarrow \mathcal{L}_{\mathrm{distill}}(\boldsymbol{z}, \mathrm{sg}(\boldsymbol{a}))$ ▷ Equation equation 8
  $\boldsymbol{g}_{\mathrm{t}}, \boldsymbol{g}_{\mathrm{r}} \leftarrow \mathrm{Backward}(\mathcal{L}_{\mathrm{LM}} + \mathcal{L}_{\mathrm{distill}})$
  $\boldsymbol{\theta}_{\mathrm{t}}, \boldsymbol{\theta}_{\mathrm{r}} \leftarrow \mathrm{OptimizerStep}(\boldsymbol{\theta}_{\mathrm{t}}, \boldsymbol{g}_{\mathrm{t}}), \mathrm{OptimizerStep}(\boldsymbol{\theta}_{\mathrm{r}}, \boldsymbol{g}_{\mathrm{r}})$
  $\boldsymbol{b} \leftarrow \mathrm{Balance}(\boldsymbol{b}, \boldsymbol{n})$ ▷ Equation equation 2
end for

During training of $\mathcal{M}_{\mathrm{t}}$, we supplement BCE with a RankNet loss to improve expert ranking. For one token at one non-hash layer, let a_j now denote the gate logit produced by $\mathcal{M}_{\mathrm{t}}$ for routed expert j. Let $\mathcal{U}$ be the top-k expert set selected by the gate logits a_j. The ranking loss encourages $\mathcal{R}$ to rank these experts above the remaining experts:

$$\mathcal{U} = \mathrm{TopK}_j(a_j, k),$$
$$\mathcal{L}_{\mathrm{rank}} = \frac{1}{k(N_r - k)} \sum_{j \in \mathcal{U}} \sum_{r \notin \mathcal{U}} \log\left(1 + \exp(z_r - z_j)\right). \tag{7}$$

With ranking weight $\lambda \in [0, 1]$, the per-token, per-layer distillation objective is

$$\mathcal{L}_{\mathrm{distill}} = (1 - \lambda)\mathcal{L}_{\mathrm{BCE}} + \lambda \mathcal{L}_{\mathrm{rank}}. \tag{8}$$

We average this objective over valid tokens and the non-hash layers. Both losses use detached gate logits produced by $\mathcal{M}_{\mathrm{t}}$, so distillation updates only the standalone router.

Algorithm 1 summarizes the training procedure for $\mathcal{M}_{\mathrm{t}}$. For one token at one layer, let $\boldsymbol{a}, \boldsymbol{z} \in \mathbb{R}^{N_r}$ denote the gate logits produced by $\mathcal{M}_{\mathrm{t}}$ and the logits produced by $\mathcal{R}$, respectively. Let $\boldsymbol{n} \in \mathbb{N}^{N_r}$ denote the expert token counts over a training batch at that layer. We omit token and layer indices in the algorithm for brevity. Let $\boldsymbol{g}_{\mathrm{t}}$ and $\boldsymbol{g}_{\mathrm{r}}$ denote the gradients with respect to $\boldsymbol{\theta}_{\mathrm{t}}$ and $\boldsymbol{\theta}_{\mathrm{r}}$, respectively, and let $\boldsymbol{b}$ collect the selection biases in non-hash layers. Each $\mathrm{OptimizerStep}$ clips the gradients, updates the optimizer state and parameters, and advances the learning-rate schedule.

4 Experiments

4.1 Experimental Setup

We evaluate AhaMoE on language modeling and zero-shot downstream tasks. Our main MoE model has 2.6B total parameters and 320M activated parameters per token, with 64 routed experts and 1 shared expert per layer and top-4 routing. All models use the same 32B-token training stream from ClimbMix (Diao et al., 2025). Architectural, optimization, and evaluation details are provided in Appendix H. We also evaluate inference latency and find that the standalone router is inexpensive to execute, as detailed in Appendix B.

4.2 Main Results

We evaluate the complete AhaMoE training recipe against hidden-routed baselines without partial parameter reuse and existing pre-routed methods. *Hidden-routed* uses learned hidden-state-dependent routing in every block. *Hash-prefix* uses Hash Layers in its first 2 layers. *Hash-prefix (replay)* repeats the initial data used for source training, preserving the full model and optimizer state. The pre-routed baselines are *Hash Layers* (Roller et al., 2021), *StableMoE* (Dai et al., 2022), and *Grouter* (Xu et al., 2026).

Figure 1 compares losses on identical training batches. Hash Layers and Grouter enjoy a brief initial advantage over the hidden-routed baselines, but this advantage disappears early in training. Both subsequently remain substantially worse, indicating that a useful initial routing policy need not remain effective as the model learns. StableMoE also underperforms the hidden-routed baselines

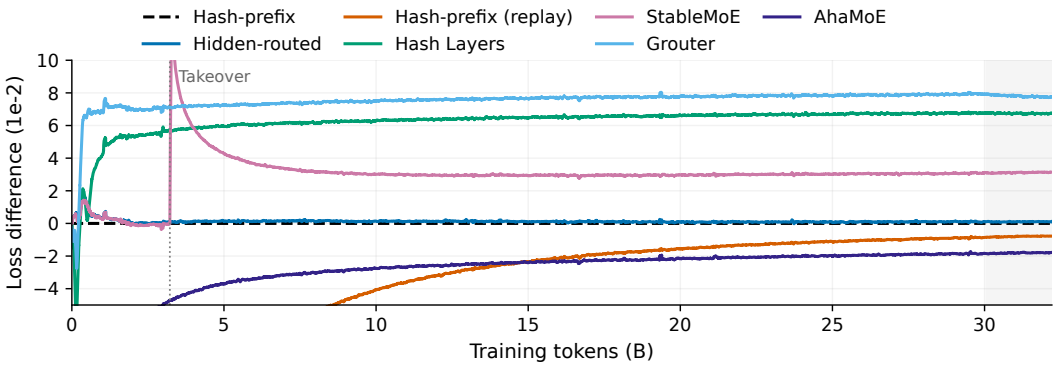

Figure 1: Training loss difference relative to Hash-prefix on matched data batches, calculated as $\mathcal{L}_{\text{model}} - \mathcal{L}_{\text{Hash-prefix}}$. Negative values indicate lower loss than Hash-prefix. Early differences outside the vertical range are clipped. The shading marks learning-rate decay.

Table 1: Zero-shot evaluation: perplexity (ppl) on WikiText (Wiki.) and LAMBADA (LMB.), and accuracy (acc/acc_n) on 7 downstream tasks. Avg. and Geo Mean. are the arithmetic and geometric means of these accuracies. Best results are **bold**, and second-best are underlined.

Model	Wiki. ppl ↓	LMB. ppl ↓	LMB. acc ↑	PIQA acc ↑	Hella. acc_n ↑	Wino. acc ↑	ARC-e acc ↑	ARC-c acc_n ↑	SIQA acc ↑	Avg. ↑	Geo Mean. ↑
Hidden-routed	21.34	17.24	**42.98**	74.59	56.35	53.99	69.28	36.18	39.66	53.29	51.57
Hash-prefix	21.23	18.38	40.87	75.14	55.93	53.43	**69.65**	**38.14**	40.28	53.35	51.66
Hash-prefix (replay)	20.98	17.91	41.22	74.92	56.17	54.30	69.40	37.37	39.51	53.27	51.53
Hash Layers	23.53	21.19	38.70	72.31	50.85	54.14	66.54	32.42	40.33	50.75	48.92
StableMoE	22.37	19.00	40.23	73.67	53.66	53.67	67.05	34.81	38.95	51.72	49.95
Grouter	23.87	21.86	39.06	72.52	50.80	51.85	65.66	33.96	37.87	50.25	48.49
AhaMoE	**20.68**	**16.19**	42.23	**75.95**	**56.86**	**56.67**	69.11	37.54	**40.58**	**54.13**	**52.42**

after its frozen standalone routers take over. The switch causes a pronounced loss increase, followed by only a partial recovery. In contrast, *AhaMoE* maintains a clear advantage over Hidden-routed and Hash-prefix throughout training. *AhaMoE* overtakes Hash-prefix (replay) later in training for two reasons: 1) partial-reuse initialization is better than simply replaying part of the training data; and 2) continual adaptation of an already initialized standalone router does not cause significant performance degradation. We will break down the performance gain in detail in Section 4.5.

Table 1 shows that the training-loss advantage transfers to downstream performance. Across all methods, AhaMoE achieves the lowest perplexity on both datasets, the highest accuracy on 4 of the 7 tasks, and the best arithmetic and geometric mean accuracies. Restricting the comparison to pre-routed models, AhaMoE is best on every benchmark. Hash-prefix (replay) improves both perplexity measures over Hash-prefix, but does not improve either aggregate accuracy. AhaMoE achieves these results without a fully trained source model. Grouter distills its router from the Hidden-routed model evaluated above, whereas AhaMoE uses a source model trained with only 13.3% of that model's training budget. This substantially reduces the source-pretraining requirement.

4.3 Expert Mixing and Continual Adaptation

All experiments in this and the following subsections use 32 routed experts per layer and train each target model on 8B tokens. Further experimental details are provided in Appendix H. We show the early and later steps independently for better display of the loss difference across steps.

The methods compared in Section 4.2 differ in router architecture and training workflow. We therefore use the same standalone router architecture and preparation procedure within each of the StableMoE-style and Grouter-style workflows to examine expert mixing and continual adaptation. Here, the routing policy is the mapping from input token sequences to expert assignments at ev-

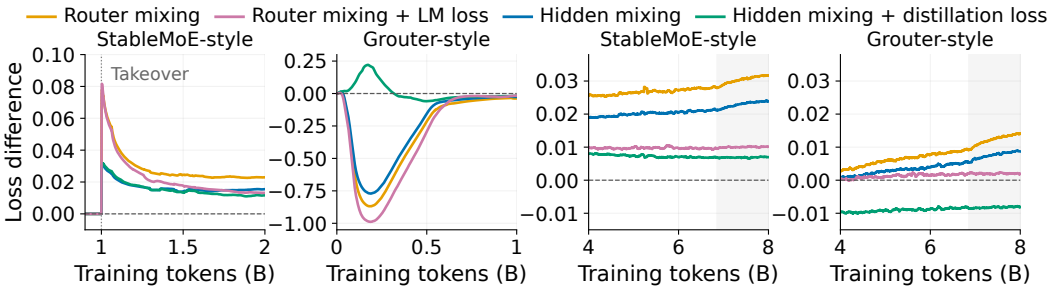

Figure 2: Effects of expert mixing and continual adaptation under StableMoE-style and Grouter-style training. Loss differences are calculated as $\mathcal{L}_{\text{model}} - \mathcal{L}_{\text{Hash-prefix}}$. The shading marks learning-rate decay.

Table 2: Ablations of AhaMoE. Differences are relative to the full recipe.

Variant	Validation loss ↓	Difference
AhaMoE: 2-block reuse + initialization + adaptation	2.28801	0.00000
Embedding only reuse	2.31921	+0.03119
Full-model reuse	2.31002	+0.02200
Without offline initialization	2.29064	+0.00262
Hidden mixing + frozen policy	2.31856	+0.03054
Router mixing + LM loss	2.31128	+0.02326
Hash-prefix trained from scratch	2.33021	+0.04220
AhaMoE with a fully trained source	2.28173	−0.00628

ery layer. *Router mixing* uses mixture weights produced by the standalone router, whereas *hidden mixing* uses weights produced by the target model's MoE gates.

Under each mixing choice, we compare a frozen routing policy with one that continues to adapt. To obtain a frozen policy, we freeze the router, its input embedding, and its selection biases. In *hidden mixing with distillation loss*, the standalone router continually learns by distilling the logits produced by the target model's MoE gates. In *router mixing with LM loss*, the language-model loss updates the standalone router through its mixture weights directly, without distillation.

Figure 2 shows that continual adaptation improves over a frozen policy in both workflows. Both hidden mixing with distillation loss and router mixing with LM loss achieve lower late-training loss than their frozen-policy counterparts. Thus, the benefit of continual adaptation holds under both sources of mixture weights. These results are consistent with our intuition from modern MoE models that routers should continually adapt to changes in the rest of the model.

Hidden mixing outperforms router mixing in both workflows, whether the routing policy is frozen or continually adapted. Grouter-style models initially favor router mixing, whereas StableMoE-style models favor hidden mixing after takeover. Despite these differences, hidden mixing with distillation loss gives the lowest late-training loss in both workflows.

4.4 Ablating the Components of AhaMoE

To examine how partial parameter reuse contributes to AhaMoE, we hold the source and target budgets fixed and vary which part of source-model parameters are reused. Table 2 shows that reusing the first 2 blocks together with the embedding layer yields lower validation loss than reusing only the embedding or the full model. The benefit therefore depends on how pretrained parameters are reused, not just on the source-training budget. We examine the effect of reuse depth in more detail in Section 4.5. We also examine router initialization and continual adaptation: removing offline initialization slightly worsens validation loss, whereas keeping the routing policy fixed throughout target training causes a much larger degradation. Initialization is beneficial, but continual adaptation has a larger effect in this setting. Among the two adaptation strategies, hidden mixing with distillation loss outperforms router mixing with LM loss under the same preparation budget. Finally, using a

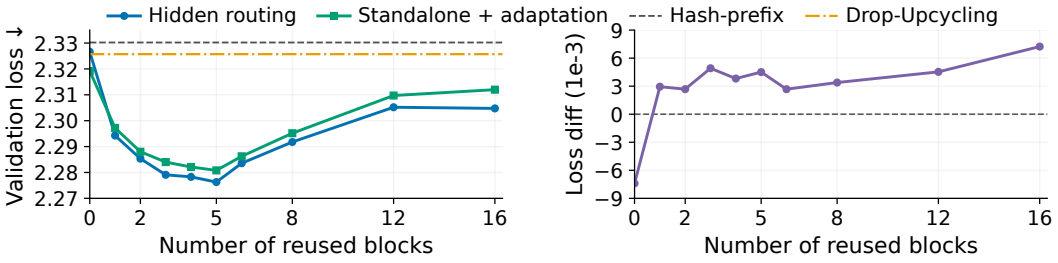

Figure 3: Effect of the number of reused layers. The left panel shows validation loss under hidden routing and a standalone router with continual adaptation. The right panel shows $\mathcal{L}_{\text{standalone}} - \mathcal{L}_{\text{hidden}}$ at each reuse depth.

fully trained source provides only a modest further improvement, indicating that the early source already delivers most of the improvement over training from scratch with a substantially smaller source-training budget.

4.5 Prefix Reuse and Its Interaction with Routing

With hidden routing, reusing the first 2 layers reduces validation loss by 0.0449 when these layers use hash layers and by 0.0430 when they use learned routers. Results with the standalone router also suggest that the reuse benefit does not depend on hashing.

We next vary the number of reused blocks from 0 to 16, with results shown in Figure 3. Note that the 16 reused layers do not include the final normalization or language-model head. Among all tested reuse depths, reusing 5 blocks yields the lowest validation loss in both routing settings. Reusing too few layers discards much of what the source model has learned, whereas reusing most layers resembles continued training on previously seen data. Repeated training on the same data can offer limited benefits and eventually lead to overfitting (Xue et al., 2023; Jha et al., 2026). Partial reuse balances knowledge retention with reinitialization, which may provide a regularizing effect analogous to dropout.

To compare with another partial-reinitialization strategy, we adapt Drop-Upcycling, which initializes a sparse MoE from a dense model while reinitializing part of each expert (Nakamura et al., 2025). We use the same source and target training budgets and match the dense model's feed-forward width to the MoE's active expert width. Even reusing a single block outperforms this baseline under both hidden routing and the standalone router. These results favor reusing only the first few Transformer layers over this dense-to-MoE conversion recipe at our experimental scale.

The reuse-depth comparison also reveals an interaction with routing. Although the standalone router improves on hidden routing with embedding-only reuse, it performs worse at every positive reuse depth tested. A possible explanation is that imperfect routing distillation creates a mismatch between the standalone router and the inherited model parameters, which can become more pronounced as more layers are reused. The layer-wise results in Appendix G.2 show that routing is learned more accurately in earlier layers, consistent with the smaller gap under shallow reuse. We therefore conservatively reuse only the first 2 hash-routed blocks in the main experiments, leaving the remaining layers free to adapt to the standalone router from random initialization.

These controlled experiments show that our partial-reuse strategy improves both hidden-routed and pre-routed models. It accounts for much of AhaMoE's quality advantage over hidden-routed models trained from scratch, while hidden-mixing with continual adaptation helps retain this advantage under pre-routing.

5 Conclusion

AhaMoE combines partial parameter reuse with continual router adaptation supervised by gates that only weight expert outputs. It enables advance expert prefetching while matching or outperforming existing hidden-routed baselines without partial parameter reuse.

AI USE STATEMENT

We used generative AI tools to support an extensive literature review to assess the novelty of our work and improve its quality. We also used these tools to draft and polish the manuscript. Under our detailed instructions, AI tools wrote code and executed the experiments we specified. We reviewed all code and verified that it followed our specifications. We take responsibility for the final content of this work, including its text, claims, code, and experimental results.

REFERENCES

Yonatan Bisk, Rowan Zellers, Ronan Le bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):7432–7439, Apr. 2020.

Ruisi Cai, Yeonju Ro, Geon-Woo Kim, Peihao Wang, Babak Ehteshami Bejnordi, Aditya Akella, and Zhangyang Wang. Read-ME : Refactorizing LLMs as router-decoupled mixture of experts with system co-design. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 116126–116148. Curran Associates, Inc., 2024. doi: 10.52202/079017-3687. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d298cf34e4539f9134db7f38b42f69fe-Paper-Conference.pdf.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. MoE-lightning: High-throughput MoE inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, ASPLOS '25, pp. 715–730, New York, NY, USA, 2025. Association for Computing Machinery.

Hongtao Chen, Weiyu Xie, Boxin Zhang, Jingqi Tang, Jiahao Wang, Jianwei Dong, Shaoyuan Chen, Ziwei Yuan, Chen Lin, Chengyu Qiu, Yuening Zhu, Qingliang Ou, Jiaqi Liao, Xianglin Chen, Zhiyuan Ai, Yongwei Wu, and Mingxing Zhang. Ktransformers: Unleashing the full potential of cpu/gpu hybrid inference for moe models. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, 2025a.

Liangkun Chen, Zijian Wen, Tian Wu, Xiaoxi Zhang, and Chuan Wu. Sp-moe: Speculative decoding and prefetching for accelerating moe-based model inference, 2025b. URL https://arxiv.org/abs/2510.10302.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge, 2018.

Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. StableMoE: Stable routing strategy for mixture of experts. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7085–7095, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.489. URL https://aclanthology.org/2022.acl-long.489/.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models, 2024. URL https://arxiv.org/abs/2401.06066.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, et al. Deepseek-v3 technical report, 2025a. URL https://arxiv.org/abs/2412.19437.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, et al. Deepseek-v3.2: Pushing the frontier of open large language models, 2025b. URL https://arxiv.org/abs/2512.02556.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, et al. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

Shizhe Diao, Yu Yang, Yonggan Fu, Xin Dong, Dan Su, Markus Kliegl, Zijia Chen, Peter Belcak, Yoshi Suhara, Hongxu Yin, Mostofa Patwary, Yingyan Lin, Jan Kautz, and Pavlo Molchanov. Nemotron-climb: Clustering-based iterative data mixture bootstrapping for language model pre-training, 2025. URL https://arxiv.org/abs/2504.13161.

Truong Giang Do, Le Khiem, Quang Pham, TrungTin Nguyen, Thanh-Nam Doan, Binh Nguyen, Chenghao Liu, Savitha Ramasamy, Xiaoli Li, and Steven Hoi. HyperRouter: Towards efficient training and inference of sparse mixture of experts. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 5754–5765, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.351. URL https://aclanthology.org/2023.emnlp-main.351/.

Daize Dong, Junlin Chen, Haolong Jia, Jiang Liu, Jiawei Wu, Huanwei Di, Jialian Wu, Zhengzhong Liu, Zicheng Liu, Emad Barsoum, Dimitris N. Metaxas, and Hongyi Wang. Pr2: Predictive routing replay for moe-based llm reinforcement learning, 2026. URL https://arxiv.org/abs/2606.00395.

Zhiyuan Fang, Yuegui Huang, Zicong Hong, Yufeng Lyu, Wuhui Chen, Yue Yu, Fan Yu, and Zibin Zheng. Klotski: Efficient mixture-of-expert inference via expert-aware multi-batch pipeline. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 574–588, New York, NY, USA, 2025. Association for Computing Machinery.

Zhiyuan Fang, Xingfan Yu, Yuegui Huang, Zicong Hong, Yufeng Lyu, Wuhui Chen, Yue Yu, and Fan Yu. Fate: Fasss sesdge inference of mixture-of-experts models via cross-layer gate. In *Proceedings of the ACM Web Conference 2026*, WWW '26, pp. 5537–5546, New York, NY, USA, 2026. Association for Computing Machinery.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, et al. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Zijin Gu, Tatiana Likhomanenko, Vimal Thilak, Jason Ramapuram, and Navdeep Jaitly. Path-constrained mixture-of-experts, 2026. URL https://arxiv.org/abs/2603.18297.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ISCA '24, pp. 1018–1031. IEEE Press, 2025.

Atindra Jha, Margaret Li, Jure Leskovec, Percy Liang, and Luke Zettlemoyer. Data scarcity and model sparsity: Mixtures-of-experts overfit more to repeated data, 2026. URL https://arxiv.org/abs/2609.11917.

Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=N5fVv6PZGz.

Byeongju Kim, Jungwan Lee, Donghyeon Han, Hoi-Jun Yoo, and Sangyeob Kim. Flashmoe: Reducing ssd i/o bottlenecks via ml-based cache replacement for mixture-of-experts inference on edge devices, 2026. URL https://arxiv.org/abs/2601.17063.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Taishi Nakamura, Takuya Akiba, Kazuki Fujii, Yusuke Oda, Rio Yokota, and Jun Suzuki. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=gx1wHnf5Vp.

Nam V. Nguyen, Huy Nguyen, Quang Pham, Van Nguyen, Savitha Ramasamy, and Nhat Ho. Competesmoe – statistically guaranteed mixture of experts training via competition, 2025. URL https://arxiv.org/abs/2505.13380.

Cicero Nogueira dos Santos, James Lee-Thorp, Isaac Noble, Chung-Ching Chang, and David Uthus. Memory augmented language models through mixture of word experts. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 4425–4438, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.249. URL https://aclanthology.org/2024.naacl-long.249/.

Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144/.

Stephen Roller, Sainbayar Sukhbaatar, Arthur Szlam, and Jason E Weston. Hash layers for large sparse models. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=lMgDDWb1ULW.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial winograd schema challenge at scale. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):8732–8740, Apr. 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4463–4473, Hong Kong, China, November 2019. Association for Computational Linguistics.

Oliver Sieberling, Bharat Runwal, Rameswar Panda, and Yoon Kim. Dynamic short convolutions improve transformers, 2026. URL https://arxiv.org/abs/2606.03825.

Jianlin Su. Where does the tid2eid in deepseek v4 come from?, May 2026. URL https://kexue.fm/archives/11750.

Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, Chao Li, and Minyi Guo. Hobbit: A mixed precision expert offloading system for fast moe inference, 2024. URL https://arxiv.org/abs/2411.01433.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, et al. Kimi linear: An expressive, efficient attention architecture, 2025. URL https://arxiv.org/abs/2510.26692.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, et al. Kimi k3: Open frontier intelligence, 2026a. URL https://arxiv.org/abs/2607.24653.

Kimi Team, Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, et al. Attention residuals, 2026b. URL https://arxiv.org/abs/2603.15031.

Wenfeng Wang, Jiacheng Liu, Xiaofeng Hou, Xinfeng Xia, Peng Tang, Mingxuan Zhang, Chao Li, and Minyi Guo. Moe-speq: Speculative quantized decoding with proactive expert prefetching and offloading for mixture-of-experts, 2025a. URL https://arxiv.org/abs/2511.14102.

Zhibin Wang, Zhonghui Zhang, Yuhang Zhou, Zibo Wang, Mo Zhou, Peng Jiang, Weilin Cai, Chengying Huan, Rong Gu, Sheng Zhong, and Chen Tian. Accelerating mixture-of-experts inference by hiding offloading latency with speculative decoding, 2025b. URL https://arxiv.org/abs/2508.21706.

Yuqi Xu, Rizhen Hu, Zihan Liu, Mou Sun, and Kun Yuan. Grouter: Decoupling routing from representation for accelerated moe training. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ztVp5EqREE.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling LLM under token-crisis. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=Af5GvIj3T5.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. Moe-infinity: Efficient moe inference on personal machines with sparsity-aware expert cache, 2025. URL https://arxiv.org/abs/2401.14361.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 4791–4800, Florence, Italy, July 2019. Association for Computational Linguistics.

Shien Zhu, Samuel Bohl, Robin Oester, and Gustavo Alonso. Pre-attention expert prediction and prefetching for mixture-of-experts large language models, 2025. URL https://arxiv.org/abs/2511.10676.

A Related Work

Hash Layers (Roller et al., 2021), StableMoE (Dai et al., 2022), and Grouter (Xu et al., 2026), reviewed in Section 2.2, determine expert assignments from input tokens. Read-ME (Cai et al., 2024) uses a causal router shared across layers and jointly trains it with experts extracted from a pretrained dense model. Its construction assigns one routed expert to each predefined data domain, with only 7 domains used in its experiments. Modern fine-grained MoE models instead distribute capacity across many smaller experts, reaching hundreds of routed experts per layer (Dai et al., 2024; DeepSeek-AI et al., 2025a). Our main experiments also use 64 routed experts per layer. Scaling Read-ME to such expert counts would require correspondingly fine-grained data partitions, making it difficult to adapt directly to modern fine-grained MoE.

MoWE (Nogueira dos Santos et al., 2024) uses fixed lexical routing with longest-match lookup over an auxiliary vocabulary constructed from Wikidata names and filtered by C4 word frequencies. This introduces an external lexical resource and explicit vocabulary-construction and matching rules, while the resulting routing policy remains fixed as the model learns. Its reliance on an externally constructed lexical vocabulary and language-specific preprocessing can complicate adaptation to multilingual settings. Hash Layers also explores fixed local context, but finds that bigram hashing underperforms current-token hashing in its experiments (Roller et al., 2021).

Other methods change how routers are parameterized or supervised. CompeteSMoE (Nguyen et al., 2025) trains a router to imitate expert-response scores obtained by occasionally evaluating all experts. HyperRouter (Do et al., 2023) generates routing parameters from trainable layer embeddings through a fixed hypernetwork. PathMoE (Gu et al., 2026) shares router parameters across consecutive layers to constrain expert paths. PR^2 (Dong et al., 2026) learns logit corrections that anticipate router evolution during reinforcement learning and replays the predicted expert indices during policy updates. All these methods still use intermediate hidden states to select experts when generating new tokens, so they do not make assignments for all layers available before model computation begins.

For hidden-routed MoE models, inference systems address the cost of expert offloading through heterogeneous execution, caching, and prefetching. KTransformers (Chen et al., 2025a) and Fiddler (Kamahori et al., 2025) use CPU–GPU execution to reduce expert-transfer overhead, while FlashMoE (Kim et al., 2026) learns a cache replacement policy for SSD-offloaded experts. MoE-Lightning (Cao et al., 2025) prefetches pages of next-layer weights while overlapping CPU attention and GPU computation across micro-batches. SpecMoEOff (Wang et al., 2025b) combines speculative decoding with CPU–GPU scheduling to amortize expert transfers over multiple tokens during verification.

To begin prefetching before routing decisions are available, several systems predict expert selections. MoE-Infinity (Xue et al., 2025) matches observed expert activations against historical request traces to predict which experts to prefetch and retain in the cache. Klotski (Fang et al., 2025) uses correlations between expert selections across layers to predict frequently used experts and prefetch them within a pipeline that processes multiple batches. FATE (Fang et al., 2026) applies the next layer's gate to the current layer's gate input to predict experts for prefetching. HOBBIT (Tang et al., 2024) applies gates from later layers to the current hidden state and combines predictions with cache information and mixed-precision loading to guide prefetching. Pre-Attention Expert Prediction (Zhu et al., 2025) trains lightweight predictors on activations before each layer's attention computation to prefetch experts for that layer. SP-MoE (Chen et al., 2025b) applies the target model's gates to draft-model attention outputs and asynchronously prefetches predicted experts during drafting. MoE-SpeQ (Wang et al., 2025a) uses expert selections from a quantized draft model to guide prefetching and cache management for full-precision verification. These systems accelerate inference for pretrained hidden-routed MoE models.

Pre-Gated MoE (Hwang et al., 2025) is a related algorithm–system co-design that fine-tunes gates to select the next MoE block's experts from earlier hidden states, enabling prefetching one block ahead. AhaMoE instead treats the availability of expert assignments for all layers before model computation as a requirement for model design. We develop a training recipe that maintains this property from the start of target-model training, making the resulting models easier to support with expert prefetching.

The initialization in AhaMoE is related to Drop-Upcycling (Nakamura et al., 2025), which initializes an MoE model from a pretrained dense model and reinitializes selected intermediate channels within each expert to promote specialization. AhaMoE instead initializes from a source MoE model, retaining its embedding and first few complete Transformer layers while reinitializing all subsequent layers. Thus, the two methods apply partial reinitialization at different granularities, within individual experts and across model depth, respectively.

AhaMoE's continual distillation also has a parallel in sparse attention. DeepSeek Sparse Attention (DeepSeek-AI et al., 2025b), uses a lightweight indexer to select key-value tokens for attention. During sparse training, the language-model loss trains the main attention module, including its query and key projections, while a separate KL divergence loss continually distills its attention distributions into the indexer. DSA distills attention distributions, whereas AhaMoE distills logits produced by MoE gates into the standalone router.

B Inference Latency

We compare the latency of computing expert assignments on the CPU with that of executing the first 2 model layers on the GPU. Both measurements use the final checkpoint of the main AhaMoE model and are performed on the same machine, equipped with Intel Xeon Platinum 8558 CPUs and an NVIDIA RTX PRO 6000 Blackwell Server Edition GPU. Each measurement processes 1 new token per sequence at decode batch sizes 1, 2, 4, and 8; reported latencies cover the entire batch. The CPU and GPU components are timed separately.

The CPU router uses 8 physical cores within a single socket, with threads and memory bound to the same non-uniform memory access (NUMA) node. An optimized C++ executor uses AVX-512 operations, materialized input embeddings, preallocated buffers, and a persistent thread pool. Weights, activations, and recurrent states are FP32. Each call executes all 8 KDA/SwiGLU router layers, updates the recurrent and convolution states, and computes logits and bias-corrected top-4 expert assignments for the remaining 14 model layers. The first 2 layers use fixed hash assignments, so their unused learned readouts and hash-table lookups are excluded from CPU router timing. The CPU measurements include the Python-to-C++ call and return of the routing results.

On the GPU, we use BF16 weights, activations, and key-value caches, with CUDA Graph replay. The graph includes the embedding lookup, hash routing, and the first 2 complete Transformer layers, including attention, routed and shared experts, hidden-state mixture weights, normalization, and residual connections. Each sequence has a key-value cache for 4,096 preceding tokens and processes 1 new query token using PyTorch's Flash implementation of scaled dot-product attention. GPU latency is measured with CUDA events around graph replay, without per-layer timing hooks and with the input-buffer copy outside the timed interval.

Both benchmarks use seeded random token IDs and measure warm decoding with resident weights. For the CPU, we pool 2,560 wall-clock samples from 2 independent runs, each with 512 warmup steps, and report the median (P50) and 95th percentile (P95). Recurrent state is retained across decode steps. For the GPU, we report P50 over 200 replay measurements after warmup. Model loading, prefill, graph capture, and CPU–GPU transfers are outside the measured intervals.

Table 3: Decode latency in milliseconds. CPU measurements cover the standalone router; GPU measurements cover the first 2 model layers.

Decode batch	CPU router (8 cores)		GPU first 2 layers
	P50	P95	P50
1	0.308	0.317	0.348
2	0.355	0.361	0.482
4	0.445	0.457	0.577
8	0.642	0.689	0.743

Table 3 shows that the CPU router's P95 is below the GPU's P50 at every tested batch size. These measurements suggest that 8 CPU cores can typically produce all learned expert assignments before the GPU finishes the first 2 layers. Because these layers use fixed hash assignments, their execution

can begin while the CPU computes routing for the remaining layers. This provides an opportunity to hide router computation behind the initial GPU layers and make subsequent expert requests available ahead of execution. These component measurements indicate the potential for overlap, whose realization in an offloading runtime also depends on scheduling and communication overheads.

C Adaptation of Hash Layers

The basic Hash Layers rule assigns each token type to a single expert (Roller et al., 2021). To use fixed token-based selection with the MoE architecture in Section 2.1, we instead construct a table H^ℓ that maps each token type to a set of k distinct routed experts at layer ℓ. For input token identity x_t, we replace the selection rule in Equation equation 1 with

$$\mathcal{S}_t^\ell = H^\ell(x_t). \tag{9}$$

The selected experts retain the normalized hidden-state affinity weights in Equation equation 1, and the shared experts are unchanged. Since the table fixes expert selection, these layers do not update a balancing bias.

Constructing a fixed table. Let V be the number of token types and $q_v > 0$ the normalized frequency of type $v \in \{1, \dots, V\}$, with $\sum_{v=1}^{V} q_v = 1$. For one layer, we omit the superscript ℓ and maintain a load estimate f_j for each expert $j \in \{1, \dots, N_r\}$, initially zero. Assigning token type v to k experts contributes q_v/k to each selected expert's load. Thus, the final loads sum to one, with a balanced target of $1/N_r$ per expert.

Algorithm 2 processes token types in descending frequency order. For each type, it computes the positive remaining load budgets r_j and converts them into sampling probabilities π_j, with a probability floor ϵ so that every expert remains eligible. It samples k distinct experts without replacement using that distribution, and only then updates their loads. The floor is applied to the normalized sampling probabilities, not to the unnormalized budgets.

Algorithm 2 Frequency-aware construction of a fixed top-k routing table

Require: Frequencies $(q_v)_{v=1}^{V}$; N_r experts; $1 \le k \le N_r$;
probability floor $0 < \epsilon < 1/N_r$; random seed s
Ensure: A table H assigning each token type to k distinct experts

```
Initialize a pseudorandom generator with seed s
f_j ← 0 for all j ∈ {1, ..., N_r}
for each token type v in nonincreasing order of q_v do
    r_j ← max{1/N_r − f_j, 0} for all j ∈ {1, ..., N_r}
    π_j ← ε + (1 − N_r ε) r_j / Σ_{a=1}^{N_r} r_a for all j ∈ {1, ..., N_r}
    Sample H(v): k distinct experts without replacement using (π_j)_{j=1}^{N_r}
    for each j ∈ H(v) do
        f_j ← f_j + q_v/k
    end for
end for
return H
```

We use $\epsilon = 10^{-6}$ and construct separate tables for different layers using independent random seeds. Randomness is used only during table construction; every subsequent occurrence of a token type uses the same stored expert set within a layer. The remaining budgets encourage balanced assignments but do not guarantee exact balance, since a token type's frequency cannot be split across occurrences and sampling is discrete.

D ADAPTION OF STABLEMOE

We retain StableMoE's two-stage procedure: the student learns from hidden-state routing in the first stage, then takes over expert selection and remains frozen in the second stage (Dai et al., 2022). We adapt three components to our baseline.

Router architecture. We use a separate token-embedding table and expert-score projection for each MoE layer. Each student router reads only the current token ID, and its parameters are not shared across layers. This follows the use of separate routers for MoE layers in StableMoE's scattered-layer setting (Dai et al., 2022).

Load balancing. StableMoE uses an auxiliary balancing loss in the first training stage. We replace it with the loss-free balancing rule in Equation equation 2. The balancing biases affect expert selection but not the mixture weights used as distillation targets.

Distillation loss. The original objective is cross-entropy with the selected expert ID as a single hard label. For one token at one layer, we instead use the k experts selected by the hidden-state gate's bias-corrected affinities $s_j + b_j$ in Equation equation 1. Their normalized mixture weights p_j define the target probabilities. Let $\boldsymbol{z} \in \mathbb{R}^{N_r}$ denote the student logits, with scalar component z_j for expert j. The adapted loss is

$$\mathcal{L}_{\text{StableMoE}} = -\sum_{j \in \mathcal{S}} p_j \log \frac{\exp(z_j)}{\sum_{r=1}^{N_r} \exp(z_r)}. \tag{10}$$

The target probabilities are detached from gradient computation, and the loss is averaged over tokens and MoE layers. When $k = 1$, the selected expert has target probability one, so this reduces to the original hard-label cross-entropy loss.

E ADAPTATION OF GROUTER

Router architecture and source model. We replace Grouter's bidirectional Transformer encoder with a causal Transformer decoder, preventing future-information leakage during evaluation and supporting autoregressive generation. The router has 3 layers with hidden size 512, 8 attention heads of dimension 64, and SwiGLU intermediate size 2,736. It uses a separate trainable token embedding table, rotary position embeddings, RMSNorm, and a bias-free expert-score projection. The source is the fully hidden-routed MoE evaluated in the main experiments, trained on the same ordered ClimbMix data and with the same token budget as the target model. Thus, source training introduces no additional unique training data.

Distillation. We retain supervision from only the source model's first MoE layer and a single router shared across all target MoE layers. For one token, let $s_j = \sigma(a_j)$ and $\hat{s}_j = \sigma(z_j)$ denote the source and student affinities for routed expert j. We replace the original softmax KL objective with Bernoulli KL divergence,

$$\mathcal{L}_{\text{Grouter}} = \frac{1}{N_r} \sum_{j=1}^{N_r} \left[s_j \log \frac{s_j}{\hat{s}_j} + (1 - s_j) \log \frac{1 - s_j}{1 - \hat{s}_j} \right]. \tag{11}$$

The source affinities are detached, and the loss is averaged over valid tokens. We implement this objective using soft-label BCE, which differs only by a term independent of the student parameters and yields the same gradients. For the main experiments, distillation uses the first 8,192 training batches, each containing 128 sequences of length 4,096. We use AdamW with learning rate 3×10^{-4}, a 1,024-update warmup followed by a constant learning rate, and the remaining optimizer settings in Appendix H.

Calibration and target training. We replace calibration through an auxiliary balancing loss with quantile balancing (Team et al., 2026a), following Appendix F for the single shared router. The calibration pool consists of the next 100 training batches, numbered 8,193–8,292, containing 52,428,800

tokens. We use 16,384 histogram bins per expert, at most 128 bias-update iterations, a relative load-error tolerance of 0.01, and patience of 24 iterations. We retain the evaluated bias vector with the smallest maximum absolute relative load error; calibration leaves the learned router parameters fixed. The calibrated biases determine top-k selection, while the selected experts' uncorrected sigmoid affinities are normalized to form mixture weights. These selections and weights are shared across all target MoE layers. The target model is randomly initialized, and the router, its private embedding, and its selection biases remain frozen throughout target training. The target model has no hidden-state gates for expert selection or mixing.

F Quantile Balancing for Router Initialization

After distillation, we freeze the standalone router and calibrate its selection biases using quantile balancing (Team et al., 2026a). Calibration uses a fixed pool drawn from the training data and does not use validation data. We describe one layer, with M calibration tokens, N_r routed experts, and $k < N_r$ selected experts per token. Let $\hat{s}_{ij}$ be the frozen router's sigmoid affinity for expert j on calibration token i, and initialize each bias b_j to zero. The desired load is Mk/N_r assignments per expert.

At each iteration, let α_i be the $(k+1)$-th largest bias-corrected affinity for token i. Holding these token thresholds fixed, we obtain a candidate bias from the $(1 - k/N_r)$-quantile of each expert's affinity margins across the calibration pool:

$$\begin{aligned} \alpha_i &= \mathrm{kth}_{k+1}\Big(\{\hat{s}_{ij} + b_j\}_{j=1}^{N_r}\Big), \\ \widetilde{b}_j &= -\operatorname{Quantile}_{1-k/N_r}\big(\{\hat{s}_{ij} - \alpha_i\}_{i=1}^{M}\big), \\ b_j^{\mathrm{new}} &= \widetilde{b}_j - \frac{1}{N_r}\sum_{r=1}^{N_r}\widetilde{b}_r. \end{aligned} \tag{12}$$

Here, kth_{k+1} returns the $(k+1)$-th largest value. The final centering step changes all expert scores by the same amount and therefore preserves their ordering. We recompute the token thresholds over the entire calibration pool at every iteration.

The implementation estimates quantiles using pooled histograms with B_{hist} bins per expert. A first pass determines global margin ranges and counts expert assignments; a second pass accumulates margin histograms and sums their counts across workers. Quantiles are recovered from the pooled counts with interpolation within the selected bin, rather than by averaging local quantiles.

Let n_j denote the observed calibration load of expert j. We measure imbalance by $\max_j |n_j/(Mk/N_r) - 1|$ and stop when it falls below a tolerance ϵ_{bal}, when it fails to improve for P_{bal} iterations, or when the iteration limit I_{bal} is reached. We retain the evaluated bias vector with the smallest imbalance. This procedure aims to improve balance but does not guarantee exactly equal loads. For AhaMoE, only biases for the non-hash layers are installed in the target model; hash-based assignments remain fixed. For Grouter, a single calibrated bias vector is shared across all target MoE layers.

G Standalone Router Architecture

G.1 Architecture Details

The standalone router has $L_r = 8$ layers with hidden size $d_r = 256$. The input projection is bias-free and maps the MoE model's embedding dimension to d_r. The stop-gradient operation prevents the router loss from updating the MoE model's embedding table, while allowing the projection to learn. Once the embedding and projection are fixed, they can be combined into a standalone embedding table for inference. The router's non-embedding parameter count is measured after this folding and excludes both the embedding table and the input projection; the projection remains trainable during training.

Each router block contains a KDA sublayer followed by a SwiGLU sublayer. The KDA sublayer projects normalized inputs to queries, keys, and values and applies a causal dynamic convolution (Sieberling et al., 2026) of kernel length 4 and dynamic rank 15 to each projection. Its 4

heads have query, key, and value dimensions of 128, giving an inner dimension of 512. Queries and keys are normalized before the recurrent update. Head outputs undergo gated RMSNorm and are projected back to hidden size 256. The SwiGLU sublayer has intermediate dimension 512.

We use Attention Residuals (Team et al., 2026b) at sublayer boundaries and intermediate readouts. Each aggregation uses a learned query to weight the embedding and preceding sublayer outputs at the same token position, with RMSNorm before the next transformation. After each KDA–SwiGLU block, a separate Attention Residuals aggregation combines the input embedding and all sublayer outputs up to that block to produce the router layer's hidden state. Each MoE layer in the assigned group uses its own bias-free linear head to map this hidden state to expert logits.

G.2 Controlled Experiments on Router Architectures

We compare standalone router configurations by distilling a frozen, fully hidden-routed source MoE pretrained on approximately 11 billion ClimbMix tokens. Router-training budgets are displayed as 1, 2, 4, and 8 billion tokens. We measure Bernoulli Kullback–Leibler (KL) divergence between source and router expert affinities on a common held-out set; lower KL indicates better imitation of the source gates. This subsection describes the setup and analyzes the sequence-modeling, depth–width, and readout comparisons in Figure 4.

Source model and data. The source model has 16 Transformer layers, hidden dimension 1024, eight attention heads of dimension 128, and a vocabulary of 32,000 tokens. Each MoE layer contains one shared expert and 32 routed experts, with four routed experts activated per token and a SwiGLU intermediate dimension of 768 per expert. All 16 layers use learned hidden-state routing with sigmoid affinities and loss-free balancing; this architecture-study source has no hash prefix. We use the source checkpoint after 20,480 optimizer steps, corresponding to 10,737,418,240 input tokens from the fixed ClimbMix training stream. Its learning-rate schedule has 1,024 warmup updates, a constant-rate phase through update 18,432, and a final 2,048-update decay proportional to $1 - \sqrt{u}$, where $u \in [0, 1]$ is progress through the decay phase. The source model and its input embedding are frozen throughout router distillation and evaluation. Distillation targets are the pre-bias gate logits from all 16 layers. This study is separate from the hash-prefix source and 14-layer distillation used in the final AhaMoE method.

Each global batch contains 128 sequences of length 4096, or 524,288 input tokens. Router training uses batches 1–16,384 of the same fixed stream. Table 4 gives the evaluated checkpoints and their exact data budgets. The validation split comprises batches 20,481–20,992: 512 batches containing 268,435,456 valid input-token positions. This split is disjoint from both the source-model and router training ranges. This is batch-level separation in the fixed data stream, not a claim of additional document-level deduplication. Reported metrics exclude masked positions and are weighted by the number of valid tokens.

Table 4: Router-distillation budgets. Figure 4 uses approximate tick labels but plots every point at its exact token coordinate.

Router updates	Input tokens	Displayed tick (billion)
2,048	1,073,741,824	1
4,096	2,147,483,648	2
8,192	4,294,967,296	4
16,384	8,589,934,592	8

Architecture comparisons. We compare 4 encoders: token-wise feed-forward networks (FFNs), denoted *Embedding*; dynamic convolution, denoted *Conv*; Kimi Delta Attention (KDA) (Team et al., 2025); and causal softmax attention, denoted *Attention*. The labels D and W denote router depth (number of blocks) and residual width, respectively. For context-dependent predictions, the *Grouped* readout scheme maps hidden states from successively deeper router layers to expert logits for successive groups of source MoE layers using separate linear heads. The *Final* scheme applies these heads to the final router layer's hidden state. Figure 4 compares the 10 distinct architectures listed in Table 5. The grouped 8-block, width-256 KDA router is reused as a reference across panels, rather

than trained separately for each comparison. All routers share the same frozen source embedding and use a trainable input projection and 16 independent expert-score heads. In both the Grouped and Final schemes, the linear head for the first source MoE layer uses the projected embedding directly. Attention Residuals are retained across all variants.

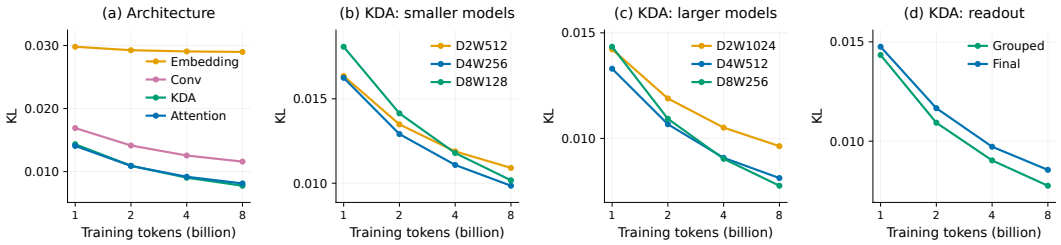

Figure 4: Router KL versus distillation data, with logarithmic horizontal axes and approximate token budgets. (a) Sequence encoders at D8W256. (b,c) KDA depth–width trade-offs at 2 scales. (d) Grouped versus final readout schemes at D8W256. Lower is better.

Table 5: Architectures plotted in the router study. D and W denote block count and residual width; FFN width is the SwiGLU intermediate dimension. Parameter counts exclude the shared frozen source embedding and include trainable projections, blocks, and heads.

Sequence mixer	D	W	Readout	FFN width	Parameters
Token-wise FFN	8	256	Grouped	800	10,242,048
Dynamic convolution	8	256	Grouped	1488	10,240,000
KDA	8	256	Grouped	512	10,216,480
Causal softmax attention	8	256	Grouped	512	8,632,320
KDA	4	512	Grouped	512	9,689,616
KDA	2	1024	Grouped	512	10,016,008
KDA	8	128	Grouped	512	6,028,320
KDA	4	256	Grouped	512	5,304,848
KDA	2	512	Grouped	512	5,238,024
KDA	8	256	Final	512	10,211,104

Optimization and objectives. All routers use AdamW with learning rate 3×10^{-4}, momentum coefficients $(0.9, 0.95)$, numerical stabilizer 10^{-8}, weight decay 0.1, and gradient clipping at norm 1. The learning rate warms up linearly over 1024 updates and remains constant thereafter, without terminal decay. Router computation uses BF16 with FP32 optimizer states, and the random seed is 42. Every curve in Figures 4 and 5 uses pure pre-bias sigmoid-affinity binary cross-entropy, with equal weighting across all 16 layers and all 32 experts.

Evaluation. We plot Bernoulli KL divergence from source affinities to router affinities, computed as the binary cross-entropy minus the source Bernoulli entropy. For each source layer, this metric is averaged over the 32 experts and valid validation tokens. The scaling figure then takes the mean over all 16 layers; the layer-wise figure retains the per-layer values. All affinities are derived from raw pre-bias logits, without load-balancing calibration. They use one training seed and do not establish statistical significance across repeated runs.

Sequence Modeling. Embedding has no sequence mixer, so we replace that sublayer with an additional SwiGLU FFN, giving it twice as many FFN sublayers as the other encoders and no cross-token computation. The convolutional variant applies a dynamic convolution (Sieberling et al., 2026) directly at the residual width, followed by an output projection. Their FFN widths approximately match the parameter count of the eight-block, width-256 KDA router. KDA and softmax attention use four heads of dimension 128, with an inner dimension of 512 and an FFN width of 512. Both apply dynamic convolutions to their query, key, and value projections; the convolutions have kernel

length four and dynamic rank 15. The softmax variant uses query/key RMSNorm and rotary position embeddings with base 500,000. All sequence mixers respect causal order and packed-document boundaries.

Under these controls, Embedding has both the highest KL and the weakest improvement with additional data (Figure 4a). Even the local sequence information captured by dynamic convolution substantially improves routing predictions, but Conv remains markedly worse than KDA and Attention, which can use long-range context. At the largest training budget, KDA achieves slightly lower KL than Attention. More importantly, for sequence length T and fixed router dimensions, KDA requires $O(T)$ sequence-processing computation and $O(1)$ recurrent-state memory during autoregressive decoding, compared with $O(T^2)$ computation and an $O(T)$ key–value cache for full attention. We therefore select KDA as the router's sequence-modeling mechanism.

Depth Modeling. At a comparable parameter budget, should a standalone router be wider or deeper? We study this trade-off at 2 scales by holding the product of depth and width fixed within each sweep. The KDA depth–width comparisons keep the attention-head and FFN dimensions fixed. The products of block count and residual width are 1024 and 2048 in the smaller and larger sweeps, respectively. The deeper configurations initially have higher KL, but improve substantially more as the training-data budget increases (Figure 4b,c). At the larger scale, the 8-block, width-256 model achieves the lowest final KL. The grouped readout scheme outperforms the final-only scheme for the 8-block router (Figure 4d). The largest reductions occur in shallow and deep source layers, as shown in Figure 5 below. These results favor allocating capacity to depth while retaining sufficient width, motivating our 8-block, width-256 router with the grouped readout scheme.

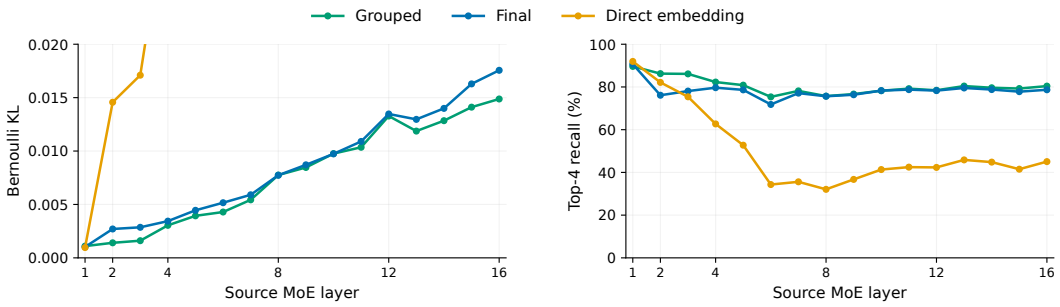

Figure 5: Layer-wise routing prediction using direct source embeddings or D8W256 KDA with grouped or final readout schemes. The left panel shows Bernoulli KL on a linear scale, with values outside the vertical range clipped; the right shows recall of the source gate's top-4 experts.

Layer-wise routing prediction. Figure 5 compares grouped and final readout schemes for the same 8-block, width-256 KDA architecture after 16,384 router updates. We also include a direct-embedding baseline that applies an independent, bias-free linear gate to the frozen source embedding for each MoE layer. This baseline has no input projection, sequence encoder, or learnable normalization; its gates are randomly initialized and trained rather than copied from the source model. All 3 variants use the same complete validation set and pure binary cross-entropy training described above. The horizontal axis indexes source MoE layers from 1 to 16, not router blocks. The grouped readout scheme gives lower KL on 13 of the 16 source layers. The largest reductions occur in shallow layers 2–3 and deep layers 15–16. The direct-embedding baseline achieves top-4 recall of 91.95% in the first layer, falling to 82.13% in the second layer. Its Bernoulli KL increases from 0.000964 to 0.014569 between these layers and grows substantially in deeper layers. In shallow layers, routing can be predicted accurately from token embeddings alone; in the first layer, this even yields better predictions than both context-dependent routers. In deeper layers, context-dependent routers perform substantially better. These results suggest that shallow-layer routing depends primarily on token identity, whereas deeper-layer routing relies more on contextual information.

H Experimental Details

H.1 Model Architecture, Data, and Optimization

The main MoE model contains 16 Transformer blocks of width 1,024, with 8 attention heads of dimension 128 and 8 key/value heads. It uses a vocabulary of 32,000 tokens, rotary position embeddings with base 500,000, and SwiGLU experts of intermediate width 768. Each MoE block has 64 routed experts and 1 shared expert; 4 routed experts are selected per token. The controlled studies retain these dimensions but use 32 routed experts per block. Each controlled target is trained for 15,360 updates, or approximately 8.05B input tokens; these experiments use separate checkpoints from the main experiments. The 4 routed experts and the shared expert together activate a feed-forward intermediate width of $5 \times 768 = 3,840$.

Training uses ClimbMix (Diao et al., 2025) with a fixed seed of 42 and the same ordered data stream for all matched targets. The global batch contains 128 sequences of length 4,096, or 524,288 input-token positions per update; the main runs use 8-way data parallelism and a local batch of 16 sequences. The 61,440-update main budget corresponds to 32,212,254,720 input-token positions, while the 15,360-update controlled-study budget corresponds to 8,053,063,680. Source training, offline distillation, and quantile-bias calibration replay portions of the training stream and never use the held-out evaluation continuation.

We use AdamW with learning rate 3×10^{-4}, momentum coefficients $(0.9, 0.95)$, numerical stabilizer 10^{-8}, weight decay 0.1, and global gradient clipping at 1.0. The learning rate warms up linearly for 1,024 updates, remains constant, and ends with a $1 - \sqrt{r}$ decay, where $r \in [0, 1]$ is progress through the final decay interval. For main targets other than Hash-prefix (replay), the flat phase ends at update 57,344 and decay lasts 4,096 updates; for controlled-study targets it ends at 13,312 and decay lasts 2,048 updates. The final learning rate is zero. The learning-rate schedules of the target model and standalone router advance synchronously. Offline router training uses a 1,024-update warmup followed by a constant learning rate. Main training uses BF16 computation.

When initializing a new target model from source weights, we do not transfer the source model's optimizer, learning-rate counter, data position, or random-number-generator state. Unless explicitly stated otherwise, target parameters outside the embedding and reused blocks are initialized afresh; reused parameters remain trainable. All targets except Hash-prefix (replay) restart their optimization and data clocks at zero. Hidden-state-dependent expert mixing remains trainable even where token-ID hashing fixes expert selection. Learned expert-selection biases use the loss-free balancing update with rate 0.001; fixed hash selection is not altered by this update.

H.2 Main Baselines and Preparation Budgets

Hash Layers uses fixed token-ID-based top-4 selection in all 16 blocks, but combines selected outputs using trainable hidden-state-dependent weights. It should not be confused with Hash-prefix, which hashes only the first 2 blocks. StableMoE uses 16 independent embedding-based student routers, one per block, rather than a shared KDA router. The students are trained alongside the MoE model until update 6,144, then frozen; update 6,145 is the first to use student expert selection. Its top-4 adaptation is described in Appendix D.

Hash-prefix (replay) first trains on batches 1–8,192, matching AhaMoE's source-training data budget. It then rewinds only the data position and trains on batches 1–61,440, retaining all model parameters and buffers, optimizer state, random-number-generator state, and learning-rate progress. There is no second warmup or parameter reinitialization. The optimizer steps run from 8,193 to 69,632, with decay over the final 4,096 updates. The first 8,192 batches are therefore presented twice, giving 36.51B token presentations over the same 32.21B-token stream.

Grouter uses a causal Transformer router distilled from the first block of a fully hidden-routed source at update 61,440. Its architecture, distillation, calibration configuration, and target-training procedure are detailed in Appendix E.

For AhaMoE, the source is Hash-prefix at update 8,192. We copy the embedding and the first 2 complete source blocks, including attention, normalization, experts, and gates, but initialize the remaining Transformer layers, final normalization, and language-model head afresh. The standalone

router has depth 8 and width 256, with the KDA architecture detailed in Appendix G.1. It is distilled with pure binary cross-entropy for the first 8,192 training batches, averaging the loss over the 14 non-hash layers. The first 2 hash layers are excluded from both offline and continual distillation. Quantile-bias calibration replays the first 512 training batches rather than continuing after batch 8,192. Calibration uses 16,384 bins, at most 128 iterations, patience 24, tolerance 0.01, and groups of 2 layers. The 512-batch data budget is distinct from the iteration budget. Its input is the current target embedding with stopped gradients. Online supervision uses 0.95 binary cross-entropy plus 0.05 RankNet loss on the 14 non-hash blocks, and the calibrated bias continues to adapt through loss-free balancing. Grouter requires 61,440 source updates, compared with 8,192 for AhaMoE: the latter is $2/15$, or 13.3%, of the former.

H.3 Evaluation and Plotting Protocol

Main benchmark results are zero-shot evaluations after the final pass through all 61,440 training batches. For Hash-prefix (replay), this corresponds to optimizer step 69,632; the other targets are evaluated at step 61,440. We report WikiText (Merity et al., 2016) and LAMBADA (Paperno et al., 2016) perplexity, and accuracy on LAMBADA, PIQA (Bisk et al., 2020), HellaSwag (Zellers et al., 2019), WinoGrande (Sakaguchi et al., 2020), ARC-Easy and ARC-Challenge (Clark et al., 2018), and SocialIQA (Sap et al., 2019). HellaSwag and ARC-Challenge use length-normalized multiple-choice accuracy; the other accuracy columns use their reported unnormalized metric. The arithmetic and geometric aggregates include exactly these 7 accuracies and exclude perplexity. Best and second-best markings are computed within each column across all 7 methods. No extra downstream fine-tuning is performed.

Controlled-study validation loss is evaluated after target update 15,360 on training-stream positions 15,361–15,872, a shared continuation of 512 batches comprising 268,435,456 input-token positions before validity masking. These batches are excluded from source training, router preparation, and target training.

Main training-loss differences use losses recorded on matching data batches, without interpolation or extrapolation. For Hash-prefix (replay), optimizer steps 8,193–69,632 align with data steps 1–61,440; its initial 8,192-update pass is excluded from the horizontal axis, as are AhaMoE's source-training and router-preparation stages. All plotted trajectories are complete over their stated intervals. Curves use a trailing 128-step mean; raw differences are not overlaid. Figure 2 shows early StableMoE-style training, early Grouter-style training, late StableMoE-style training, and late Grouter-style training from left to right. The early panels cover updates 2,048–4,096 around StableMoE takeover and updates 1–2,048 for Grouter-style training; the late panels show updates 8,192–15,360 with identical vertical limits. The reference is Hash-prefix at matched target steps, and shading marks the final learning-rate decay. For the early StableMoE-style panel, differences before update 2,048 are shown as zero, and the smoothing window starts at update 2,048, excluding all earlier differences; initial windows average only the available points, without padding. The shared initial 2,048 updates in the StableMoE-style comparison are the actual common training trajectory from which the branches resume, not independently measured duplicate runs.

H.4 Controlled Workflows and Design Ablations

The controlled router comparisons use the same depth-8, width-256 KDA architecture with separate layer-wise prediction heads, rather than the original baselines' different router architectures. Both retain a fixed hash prefix of 2 blocks. Offline distillation uses binary cross-entropy; hidden mixing with distillation loss uses 0.95 binary cross-entropy plus 0.05 RankNet on the 14 non-hash blocks. Router mixing normalizes the selected standalone sigmoid scores, whereas hidden mixing normalizes scores from the target model's gates for the selected experts.

In the StableMoE-style workflow, the MoE model and student router are jointly trained for 2,048 updates. The standalone router takes over at update 2,049. For both fixed-policy variants, its private embedding, router parameters, and selection bias are frozen at takeover. The adapting variants instead read the current target embedding with stopped gradients and continue router and balancing-bias updates. These settings compare a fixed routing policy with one that continues to adapt during training.

In the Grouter-style workflow, all target-model parameters are randomly initialized. A source trained for 15,360 updates provides targets for 8,192 offline router updates; calibration then uses batches 8,193–8,292. The fixed-policy variants freeze router parameters, the private embedding, and bias. The adapting variants also keep the source embedding as a frozen input buffer, but update the router's projection, KDA blocks, output heads, and balancing bias from the first target step. This preserves its offline-first workflow while allowing a controlled comparison of mixing and adaptation within that workflow.

For router mixing with LM loss, the router selects experts and supplies their mixture weights after takeover. The language-model loss updates the router through these weights, with no gradient through discrete expert selection and no further distillation. Each variant retains the initialization, input embedding, and balancing-bias updates of its workflow's adapting hidden-mixing counterpart. The first 2 layers retain fixed hash assignments and hidden mixing; the remaining hidden gates are not used.

The default AhaMoE design ablation uses a 4,096-update Hash-prefix source, 4,096 offline router updates, and calibration on the first 512 batches. Embedding-only and full-model reuse share this preparation, while varying parameters inherited from the source model. The full-model reuse variant includes the final normalization and language-model head. The random-router variant removes both learned offline router weights and calibrated bias initialization. The hidden-mixing variant with a frozen policy keeps the offline private embedding, router, and bias fixed during target training, while its hidden-state gates continue learning mixture weights. The router-mixing variant with LM loss uses the same early-source router initialization and reuses the same embedding and first 2 blocks as AhaMoE. From the first target update, it uses the LM-loss training procedure described above instead of hidden mixing and continual distillation. The fully trained-source variant replaces both the teacher and inherited prefix with their 15,360-update counterparts, but retains 4,096 offline updates and first-512-batch calibration.

H.5 Prefix-Depth and Dense-to-MoE Controls

For the prefix-depth sweep, the source is fixed at update 4,096. All variants copy the source embedding and the first d complete blocks for $d \in \{0, 1, 2, 3, 4, 5, 6, 8, 12, 16\}$; the final normalization and language-model head are always fresh. The number of hash-routed blocks remains 2, independently of d. In particular, at depth 1 the second block still hashes expert selection but has fresh parameters. The hidden-routing branch keeps learned hidden routing in blocks 3–16, whereas the standalone branch uses the default initialized, continually adapting router in those blocks.

For the no-hash controls, a fully hidden-routed source supplies the embedding and first 2 complete blocks at update 4,096. The hidden-routing target retains hidden routing in every block. The standalone target initializes its router from this no-hash source, predicts all 16 blocks, and continues distillation throughout target training. Each is compared with the matching fully hidden-routed from-scratch baseline, rather than with a source that used hashing.

Our Drop-Upcycling comparison is adapted to fine-grained experts and a shared expert. The dense source has feed-forward width 3,840, matching the combined intermediate width of 4 active routed experts and 1 shared expert, and trains for 4,096 updates. At conversion, each shared expert copies 768 sampled dense channels. Each routed expert independently samples 768 channels without replacement within that expert, then reinitializes 384 of them; overlaps across experts and with the shared expert are allowed. The same channel indices are used for the gate, up, and down projections, and reinitialization statistics are computed from the reset subset. The shared down-projection scale is 1 and the routed down-projection scale is 4. Non-feed-forward source parameters are copied, hidden gates are fresh, and the target retains the same 2-block hash prefix as the comparison models. The converted MoE then trains for 15,360 updates on the same target data stream. This is a matched-budget adaptation of the conversion recipe to our architecture, not a reproduction of every configuration in the original Drop-Upcycling paper.