RETHINKING MOE ROUTING: UNLOCK THE SPARSITY POTENTIAL OF PRETRAINED MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Mixture-of-experts (MoE) models expand capacity through sparse activation and are widely used in large language models. However, fixed Top-K routing assigns every token the same expert budget, creating redundant computation. Training-free skipping based on routing scores or importance estimates can compromise quality under aggressive pruning, while jointly adapting routers and experts requires optimizing a large pretrained parameter set. Our analysis reveals that pretrained MoEs can retain their capabilities under smaller expert budgets by relearning expert selection, while adaptive budgets across tokens and layers further reduce computation without sacrificing quality. Guided by these findings, we develop BEAM (Binary Expert Activation Masking), which combines hidden states with a routing prior to select which candidate experts execute. Under a shared sparsity objective, mask routers distribute a global expert budget across tokens and layers, while layer-wise self-distillation preserves the original model's representations and predictions with pretrained weights frozen. The learned policy exhibits dynamic sparsity, retaining computation for sensitive tokens and layers while pruning redundant activations elsewhere. On Qwen3-30B-A3B, BEAM preserves over 99% of the original model's performance while reducing expert activations by 50%. To deploy dynamic MoEs efficiently, we optimize BEAM kernels for high-performance inference engines. With our kernels integrated into vLLM, BEAM achieves up to $1.29\times$ throughput and $1.91\times$ TPOT speedups over the original MoE.

1 INTRODUCTION

> *"Knowledge is of two kinds. We know a subject ourselves, or we know where we can find information upon it."*
>
> —*Samuel Johnson*

Pretrained mixture-of-experts (MoE) models possess a rich collection of expert knowledge, but how efficiently that knowledge can be used depends on the routing policy (Shazeer et al., 2017; Fedus et al., 2022; Qwen Team, 2025). Yet fixed Top-K routing assigns a uniform expert budget across tokens and layers, potentially performing unnecessary computation. Further acceleration therefore calls for more selective use of pretrained expert knowledge while preserving model capabilities.

Existing methods mainly follow two routes to reduce expert activation. Training-free approaches skip expert computation using routing scores or calibrated importance estimates (Lu et al., 2024), but aggressive pruning can damage model quality. Training-based approaches adapt expert usage (Muzio et al., 2024), from learned activation counts (Yue et al., 2024) and zero-expert adaptation (Lv et al., 2026) to joint optimization of routers and experts (Jin et al., 2024; Guo et al., 2025). With the expert bank fixed, quality loss from truncating the original ranking cannot tell us whether experts lack capacity or routing allocates that capacity poorly.

To identify what prevents deeper sparsity, we conduct controlled recovery and removal-sensitivity analyses with the expert bank frozen (Section 3). We find that pretrained MoEs retain their capabilities under much smaller budgets once expert selection is relearned, showing that the main limitation lies in routing rather than insufficient expert knowledge. Native rank remains informative, but retaining its Top-K prefix is not always the best pruning decision. Moreover, pruning sensitivity varies substantially across tokens and layers. **Therefore, the core challenge is to learn which experts matter and how much computation each token and layer needs.**

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.

These findings motivate **BEAM** (Binary Expert Activation Masking), which learns binary masks over the pretrained router's candidates while freezing the original model (Section 4). For each token and layer, a lightweight mask router combines hidden states with the original routing scores to determine which candidate experts execute. A shared sparsity objective regulates the average expert budget while allowing the allocation to vary across tokens and layers. Layer-wise self-distillation then constrains the sparse execution path by matching the original model's intermediate representations and output predictions. Together, BEAM learns expert selection and budget allocation without updating pretrained weights.

The learned allocation reflects pruning sensitivity: BEAM retains more computation in layers independently identified as sensitive, without layer-specific budget targets, and varies retention across tokens and expert ranks. This supports learning sparsity from the consequences of removing computation. On Qwen3-30B-A3B, BEAM preserves over 99% of the original model's performance while reducing expert activations by 50%. BEAM can also be combined with on-policy self-distillation (OPSD) (Agarwal et al., 2024; Lv et al., 2026) to further restore performance on long-form reasoning tasks. Realizing these savings requires efficient execution of irregular assignments (Gale et al., 2023; Jin et al., 2026). We therefore develop plug-and-play dynamic MoE kernels that remove masked assignments before expert computation, delivering up to $1.29\times$ throughput and $1.91\times$ TPOT speedups in vLLM (Kwon et al., 2023).

Our main contributions can be summarized as follows:

- We expose untapped sparsity in fixed Top-K routing and show that lower budgets require relearning both expert selection and computation allocation.
- We introduce BEAM, which learns binary expert masks and dynamic budgets through a shared sparsity objective and layer-wise self-distillation while freezing pretrained weights.
- Experiments on Qwen3 and GLM validate quality retention and sensitivity-aligned allocation under reduced budgets, while optimized kernels translate expert sparsity into practical inference speedups.

2 Background and Problem Formulation

Consider a pretrained MoE with L routed layers and E experts per layer. For one token at one layer, let $\mathbf{h} \in \mathbb{R}^d$ be the normalized input to the MoE. We omit the layer and token indices in local forward equations. The router R produces logits $\mathbf{z} = R(\mathbf{h})$ and a candidate set $C = \mathrm{TopK}_{K_0}(\mathbf{z})$ (Shazeer et al., 2017). With expert functions E_e, the candidate weights and MoE output are:

$$p_e = \frac{\exp z_e}{\sum_{j \in C} \exp z_j} \quad (e \in C), \qquad \mathbf{y} = \sum_{e \in C} p_e E_e(\mathbf{h}). \tag{1}$$

We set $p_e = 0$ outside C, so $\mathbf{p} \in \mathbb{R}^E$ records the original mixture coefficients on the expert axis. The surrounding attention, normalization, and residual operations follow the original architecture.

We seek a binary vector $\mathbf{m}_i \in \{0,1\}^E$, supported on C_i, at each token–layer site i. For the set $\mathcal{I}$ of valid sites, the average activation count and sparsity are

$$\bar{K} = \frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \sum_{e \in C_i} m_{i,e}, \qquad \rho_{\mathrm{act}} = 1 - \frac{\bar{K}}{K_0}. \tag{2}$$

Fixed Top-K routing uses the same number of experts at every token–layer site. Dynamic routing instead allows this number to depend on the input and layer, allocating a shared computation budget according to the needs of each context. The objective is to preserve the pretrained model's capabilities with fewer expert activations on average.

3 Identifying Removable Expert Computation

We analyze the sparsity potential of pretrained MoEs from three perspectives: quality recovery through routing adaptation, the relationship between routing rank and expert importance, and pruning sensitivity across layers and token positions. Together, these analyses inform how expert computation can be allocated under a reduced budget.

3.1 Can Pruning Loss Be Recovered with Frozen Experts?

We reduce native Top-8 routing to Top-4, then compare router-only and full-parameter adaptation on 8,000 training sequences. The first changes expert selection and scores while freezing expert functions; the second also updates the expert bank. Table 1 reports the resulting task scores.

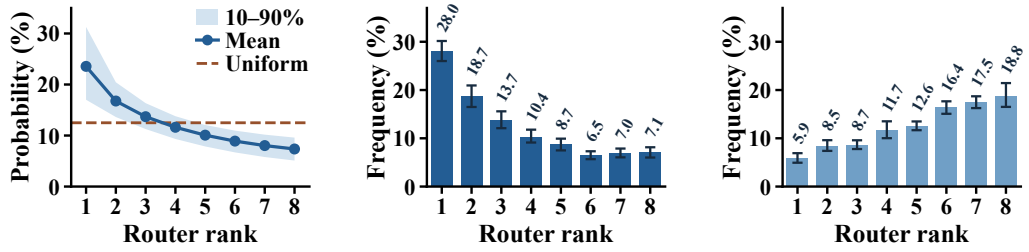
Table 1: Quality recovery after pruning.

Method	K	MATH	MMLU	Eval+	IFEval	MBPP+	Avg.
Native	8	94.80	89.93	84.76	84.84	79.10	86.69
Direct	4	93.40	87.91	78.05	72.46	72.49	80.86
Router only	4	93.60	88.08	81.10	76.89	73.28	82.59
Full FT	4	94.00	88.71	73.78	80.22	73.02	81.95

Direct truncation substantially lowers the five-task average. Router-only adaptation improves every reported task and recovers part of this loss; under the tested recipe, its average is comparable to full-parameter adaptation. A gap to native routing remains. **Pruning loss is partly recoverable through routing adaptation, without changing expert weights.**

3.2 Does Router Rank Identify Which Candidates Can Be Skipped?

Native router probabilities decrease with rank, so prefix-based methods preserve the highest-ranked experts and prune later ones first (Huang et al., 2024; Yue et al., 2024). We test this assumption by removing each candidate and measuring the induced next-token KL. Figure 1 shows that rank 1 is the largest-impact removal in only 28.0% of cases, whereas ranks 5–8 collectively account for 29.3%. Lower ranks are more often safe to remove, but the overlap is substantial: a lower-ranked expert can be critical and a highly ranked expert locally redundant. **Router rank predicts removal importance on average, but retaining its Top-K prefix is not always the best pruning decision.** Candidate-wise retention can therefore depart from the native prefix while retaining routing scores as a prior. This single-removal test motivates flexibility but does not identify the optimal joint subset.

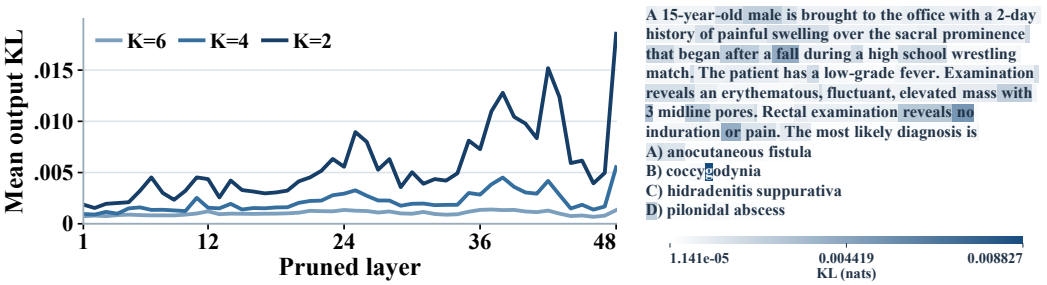


(a) Routing probability (b) Largest removal impact (c) Smallest removal impact

Figure 1: Routing probabilities and rank frequencies of the largest and smallest removal effects.

3.3 Where Can Expert Computation Be Removed?

We examine how the effect of the same pruning strength changes across layers, tokens, and positions within a sequence.



(a) Layer sensitivity (b) Token sensitivity

Figure 2: Pruning sensitivity across (a) layers and (b) tokens.

Figure 2(a) reduces all tokens in one layer to $K = 6, 4, 2$ and measures mean output KL on a fixed reasoning sequence. The early layers generally tolerate pruning better, whereas several middle and late layers exhibit pronounced sensitivity peaks; the final layer incurs the largest deviation. These differences become more pronounced as pruning becomes more aggressive.

Figure 2(b) prunes one token across all layers and measures full-prompt mean next-token KL. Removal has a much larger effect at a small number of positions, including the negation "no," than at many neighboring tokens; even repeated occurrences of "a" exhibit different sensitivities. Figure 3 further localizes the intervention to one token–layer site in a reasoning trace and measures current next-token KL, revealing that sensitive token positions also differ across layers. **The budget needed to preserve model behavior is context-dependent: it varies across layers, across tokens, and across occurrences of the same token.** This heterogeneity motivates learning where to retain computation under a shared budget, which we develop next.

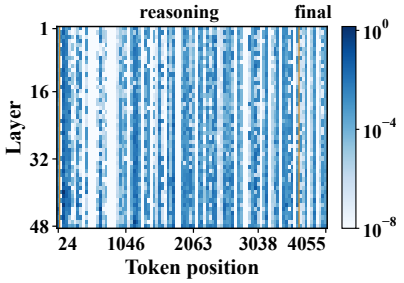


Figure 3: Token–layer sensitivity (next-token KL) under Top-8 to Top-4 pruning.

4 BEAM: Learning to Retain Expert Computation

Building on the routing analysis in Section 3, BEAM learns binary masks over the pretrained router's candidates to enable context-dependent expert activation (Figure 4). We first describe the mask router, then introduce its sparsity-guided self-distillation objective and efficient inference implementation.

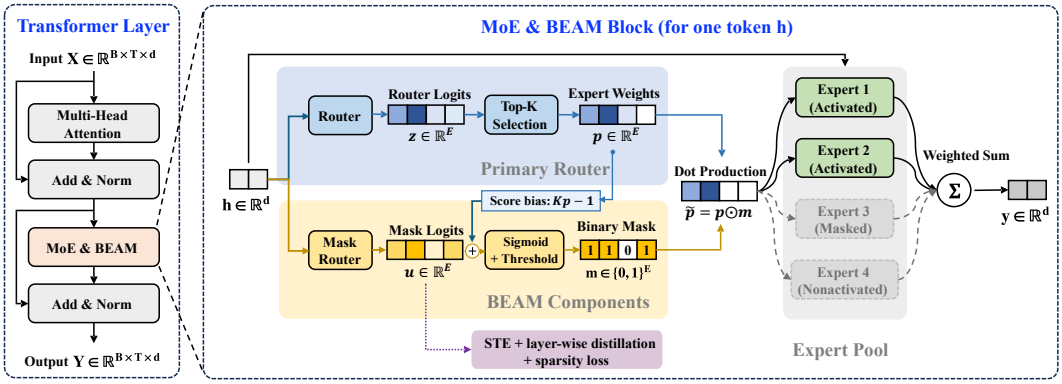


Figure 4: Overview of BEAM.

4.1 Binary Expert Activation Masking

For one token at one layer, let $\mathbf{h}$ denote the normalized pre-MoE hidden state. Each layer has its own mask gate; token and layer indices are omitted below.

Step 1: Standard routing. The recovery observed in Section 3.1 motivates adapting expert usage while retaining the pretrained expert functions. The frozen router computes logits $\mathbf{z} = R(\mathbf{h})$, proposes the Top-$K_0$ candidate set C, and normalizes its mixture coefficients $\mathbf{p}$ as in Equation 1. The mask operates only within C, so the pretrained router continues to determine which expert functions are available.

Step 2: Mask generation. Section 3.2 shows that native scores are informative but do not fully determine removal cost. We therefore use them as a prior that a hidden-state-conditioned gate can refine. A linear gate with matrix $\mathbf{W}^{\mathrm{M}} \in \mathbb{R}^{E\times d}$ and bias $\mathbf{b}^{\mathrm{M}} \in \mathbb{R}^E$ predicts retention scores:

$$\mathbf{u} = \mathbf{W}^{\mathrm{M}}\,\mathrm{sg}(\mathbf{h}) + \mathbf{b}^{\mathrm{M}}, \qquad \mathbf{s}_C = \sigma(\mathbf{u}_C + \mathrm{sg}(K_0\mathbf{p}_C - \mathbf{1}_{K_0}))\,, \quad \mathbf{s}_{\bar{C}} = \mathbf{0}. \tag{3}$$

Here σ acts element-wise, sg stops gradients, and $\mathbf{s}$ is the retention-score vector. The centered routing prior averages to zero over the candidates, favoring high-weight experts while allowing the hidden-state term to adjust their retention. Initializing $\mathbf{W}^{\mathrm{M}} = \mathbf{0}_{E\times d}$ and $\mathbf{b}^{\mathrm{M}} = \mathbf{1}_E$ retains all candidates before adaptation. Table 3 compares gate inputs and evaluates the contribution of the routing prior.

Step 3: Binary retention. To accommodate the heterogeneous sensitivity identified in Section 3.3, we threshold candidates individually, allowing both the retained subset and its size to depend on context:

$$\widehat{\mathbf{m}}_C = \mathbb{I}\big[\mathbf{s}_C \geq \tfrac{1}{2}\big], \qquad \widehat{\mathbf{m}}_{\bar{C}} = \mathbf{0}. \tag{4}$$

$\mathbb{I}$ acts element-wise. The resulting vector $\widehat{\mathbf{m}}$ allows non-prefix subsets of varying size. An activation safeguard prevents undersized selections and produces the execution mask $\mathbf{m}^{\mathrm{hard}}$. We distinguish the learned threshold decisions from this safeguard in Section 6. Appendix A.4 specifies the reported checkpoint's rule and a gate-score fallback that adds candidates only when too few survive. To optimize these discrete decisions, training uses the straight-through estimator (Bengio et al., 2013)

$$\mathbf{m}^{\mathrm{ST}} = \mathrm{sg}(\mathbf{m}^{\mathrm{hard}} - \mathbf{s}) + \mathbf{s}, \qquad \mathbf{m}^{\mathrm{ST}} \overset{\mathrm{forward}}{=} \mathbf{m}^{\mathrm{hard}}. \tag{5}$$

The forward pass therefore executes a binary decision, while gradients follow the sigmoid score. We examine this training choice in the STE ablation (Table 4). The safeguard is examined in Table 3; Appendix A gives the gradient conventions.

Step 4: Masked aggregation. Let $\mathbf{m}$ denote $\mathbf{m}^{\mathrm{ST}}$ during training and $\mathbf{m}^{\mathrm{hard}}$ during inference. Let $\mathbf{F}_C(\mathbf{h}) \in \mathbb{R}^{d \times K_0}$ stack the candidate expert outputs as columns. The masked coefficients and layer output are

$$\widetilde{\mathbf{p}}_C = \mathbf{p}_C \odot \mathbf{m}_C, \qquad \mathbf{y}^{\mathrm{S}} = \mathbf{F}_C(\mathbf{h})\widetilde{\mathbf{p}}_C, \tag{6}$$

where $\odot$ denotes element-wise multiplication. The surviving weights are not renormalized. Attention, normalization, residual operations, and expert parameters remain unchanged.

4.2 Sparsity-Guided Self-Distillation

The analyses in Section 3 identify opportunities for pruning, but combining removals throughout the model can change the representations on which subsequent routing decisions depend. We therefore train the mask gates to balance a shared sparsity target against self-distillation from the original model. On offline trajectories generated by that model, the teacher retains all candidates while the student propagates its own sparse states.

Layer-wise self-distillation. To constrain deviations as they propagate through the sparse model, we supervise both intermediate representations (Romero et al., 2015) and final predictions. Let $\mathbf{H}_\ell^{\mathrm{T}}$ and $\mathbf{H}_\ell^{\mathrm{S}}$ denote teacher and student post-block hidden-state matrices. We align these states at every routed layer:

$$\mathcal{L}_{\mathrm{path}} = \frac{1}{L}\sum_{\ell=1}^{L} D(\mathbf{H}_\ell^{\mathrm{S}}, \mathrm{sg}(\mathbf{H}_\ell^{\mathrm{T}})), \qquad \mathcal{L}_{\mathrm{distill}} = \mathcal{L}_{\mathrm{path}} + \mathcal{L}_{\mathrm{out}}. \tag{7}$$

D averages token-wise squared error normalized by teacher-state energy, with a floor to stabilize small denominators. The output term $\mathcal{L}_{\mathrm{out}}$ additionally matches coarsened next-token distributions using teacher-to-student KL (Hinton et al., 2015). The student passes its own sparse states to subsequent layers, so intermediate matching constrains accumulated deviations rather than only isolated expert outputs. Exact normalization and the compact vocabulary representation are given in Appendix A.

Target-budget sparsification. The sensitivity differences in Section 3.3 motivate constraining the average budget without prescribing a count for each token or layer. Let $\mathcal{I}_n$ contain valid token–layer sites at training step n, with C_i denoting the original candidates at site i. For a target average count $K^\star$, we regulate the retained fraction through a signed budget term:

$$U_n = \frac{1}{K_0|\mathcal{I}_n|}\sum_{i\in\mathcal{I}_n} \mathbf{1}_E^\top \mathbf{m}_i^{\mathrm{ST}}, \qquad \mathcal{L}_{\mathrm{sparse}} = \lambda_n U_n. \tag{8}$$

The coefficient follows the previous step's global count: $\lambda_n = \beta\ \mathrm{sign}(\bar{K}_{n-1} - K^\star)$, with $\beta > 0$ and $\lambda_0 = 0$. Above the target, a positive coefficient discourages retention; below it, a negative coefficient encourages retention. This feedback regulates the average budget while allowing different tokens and layers to use different counts. The main setting targets half the native activation budget.

The complete objective is

$$\mathcal{L} = \mathcal{L}_{\mathrm{distill}} + \mathcal{L}_{\mathrm{sparse}} = \mathcal{L}_{\mathrm{path}} + \mathcal{L}_{\mathrm{out}} + \lambda_n U_n. \tag{9}$$

Gradients pass through the student's frozen transformations and update only the mask gates. The two distillation terms have unit weight, and no additional token cross-entropy is used. Table 4 examines hidden-state alignment and the sparsity objective; optimization details are in Appendix A.

BEAM can optionally be followed by on-policy self-distillation (OPSD), which continues mask-only training on trajectories sampled from the current sparse policy while retaining the original teacher and objectives. We use this stage to improve long-form reasoning, with details provided in Appendix A.3.

4.3 Efficient Inference

At inference, the mask router produces a deterministic binary decision for each native candidate by thresholding its retention score at 0.5 and applying the activation safeguard described in Section 4.1. Only experts selected by $\mathbf{m}^{\mathrm{hard}}$ are evaluated, and their outputs are combined with the original mixture coefficients without renormalization, as in Equation 6. This uses the same hard-mask rule as the training forward pass, with no teacher or gradient surrogate required. We provide an efficient operator implementation for inference; its integration and performance are presented in Section 5.4.

5 Experiments

We evaluate whether BEAM preserves model capabilities under reduced expert budgets, identify the design choices responsible for this recovery, and determine whether the resulting sparsity translates into practical inference speedups.

5.1 Experimental Setup

Models and data. We evaluate Qwen3-30B-A3B (Qwen Team, 2025) and GLM-4.7-Flash (GLM Team et al., 2025). Qwen3 training pairs prompts from AceReason-1.1-SFT and Llama-Nemotron-Post-Training-Dataset (Liu et al., 2025; Bercovich et al., 2025) with responses generated by the native model. These trajectories provide self-distillation targets rather than external supervision or new knowledge; GLM adaptation is detailed in Appendix A.

Tasks and baselines. We group nine benchmarks into reasoning and knowledge (MATH-500, AIME24, GPQA-Diamond, MMLU-Redux, and GSM8K), code (HumanEval+ and MBPP+), and instruction following (IFEval and IFBench). We report each task's standard metric and average retention $\mathcal{R}$, the mean task-wise score relative to the matched native model (Appendix B). Fixed-budget baselines include native Top-8, Top-4 router-only adaptation, and full-parameter adaptation; dynamic baselines include Dynamic-MoE (Huang et al., 2024), Ada-K (Yue et al., 2024), and ZEDA (Lv et al., 2026), using their published optimization procedures.

Training and inference. Qwen3 mask training updates only the mask gates for two Adam (Kingma & Ba, 2015) passes with learning rate 10^{-5}, global batch size 16, target count four, $\beta = 0.1$, and unit distillation weights. Unless stated otherwise, dynamic methods are evaluated near half of the native routed-expert budget. Base adaptation uses Relax (Zhang et al., 2026) on eight H800 GPUs, while optional OPSD uses eight H20 GPUs. Serving evaluation uses vLLM 0.29 (Kwon et al., 2023), temperature 0.6, top-p 0.95, top-k 20, and a 32K output limit.

5.2 Accuracy Evaluation

Capability preservation at a reduced budget. At roughly half of the native expert budget, base BEAM retains 97.93% of Qwen3 performance while outperforming both fixed-count adaptation baselines (Table 2). On GLM, BEAM retains 99.33% at target $K = 3$. These results support the finding in Section 3.1: the expert bank retains sufficient capability, but its use must be relearned for a smaller budget.

Comparison with dynamic routing. At a comparable budget, BEAM improves average retention over Dynamic-MoE and Ada-K because it learns candidate-wise, non-prefix retention rather than adapting only the expert count along the router ranking. Base BEAM is competitive with ZEDA, while optional OPSD raises Qwen3 retention above 99%, with its largest gains on reasoning-intensive

Table 2: Accuracy and performance retention under reduced expert budgets.

Method	K	Reasoning & Knowledge					Code		Instruction		$\mathcal{R}$
		MATH	AIME24	GPQA-D	MMLU-R	GSM8K	Eval+	MBPP+	IFEval	IFBench	
Qwen3-30B-A3B	8.00	95.20	79.58	63.13	89.84	95.00	86.59	80.42	86.69	42.33	100.00
Router only	4.00	92.00	71.67	58.08	87.82	93.71	80.49	74.60	72.83	30.00	90.63
Full Finetune	4.00	94.40	68.75	56.57	86.75	94.01	88.41	**78.84**	77.63	34.33	93.50
Dynamic-MoE	4.19	92.20	67.92	53.03	87.30	93.18	85.37	75.40	65.25	30.67	89.06
Ada-K + PPO	4.02	**94.60**	72.50	58.59	88.91	94.92	**90.85**	76.46	80.78	**40.33**	96.73
ZEDA + OPSD	3.97	**94.60**	77.08	59.60	89.15	**95.22**	90.24	**78.84**	**84.47**	39.67	98.17
BEAM	4.03	93.60	76.25	60.10	**89.29**	94.31	90.24	78.57	83.36	**40.33**	97.93
BEAM + OPSD	3.95	94.40	**79.17**	**68.18**	88.87	94.01	90.24	76.98	84.10	39.00	**99.29**
GLM-4.7-Flash	4.00	95.80	78.75	66.16	89.94	94.84	89.63	75.66	81.89	55.00	100.00
Router only	2.00	57.20	18.33	18.69	62.02	71.95	51.22	53.44	63.59	36.67	58.68
BEAM$_{K=2}$	2.00	94.60	68.75	56.06	87.90	94.31	87.80	79.37	78.56	53.00	95.90
BEAM$_{K=2}$ + OPSD	2.00	94.20	70.83	58.59	87.61	94.39	**90.24**	78.31	79.67	52.67	96.77
BEAM$_{K=3}$	3.00	96.20	73.33	59.60	**89.55**	**95.30**	89.63	**80.95**	85.58	54.33	99.33
BEAM$_{K=3}$ + OPSD	3.00	**96.40**	**75.42**	**63.13**	89.27	94.77	**90.24**	78.57	**86.88**	**55.33**	**100.25**

benchmarks. GLM exhibits the same recovery pattern and reaches native-level average quality at target $K = 3$.

5.3 Ablation Studies

Having established the overall quality gain, we first isolate the architectural choices in the mask router (Table 3). Hidden states provide the strongest overall result among the tested inputs, indicating that token context contains information unavailable from router logits or attention outputs alone. Removing the routing prior causes a smaller but consistent decline, showing that native scores remain useful as a prior. Reducing the safeguard from two candidates to one produces the largest degradation, suggesting that it stabilizes extremely sparse decisions rather than prescribing which native ranks must survive.

Table 3: Model-design ablations on Qwen3.

Design	$\bar{K}$	Reasoning & Knowledge					Code		Instruction		Avg.	ΔAvg.
		MATH	AIME24	GPQA-D	MMLU-R	GSM8K	Eval+	MBPP+	IFEval	IFBench		
BEAM (hidden state)	4.03	93.60	76.25	60.10	89.29	94.31	90.24	78.57	83.36	40.33	78.45	–
Router-logit input	4.13	93.80	70.42	62.12	88.19	93.10	90.24	77.51	81.70	39.67	77.42	-1.3%
Attention-output input	4.02	93.80	70.42	58.08	88.77	94.09	88.41	78.31	85.03	39.33	77.36	-1.4%
w/o routing prior	4.15	94.40	69.17	61.11	89.16	94.31	87.20	78.31	83.36	40.67	77.52	-1.2%
1-expert safeguard	4.02	83.00	39.17	56.57	88.44	84.46	87.20	77.25	83.18	41.33	71.18	-9.3%

We next isolate the learning objectives and discrete optimization choices (Table 4). Removing layer-wise distillation causes a small average decline with mixed task effects. The continuous soft-mask alternative, evaluated with fixed-four selection at inference, degrades reasoning performance more substantially. Removing the sparsity objective raises raw quality only because the average count returns to 7.83, and therefore no longer satisfies the target compute budget. Together, the sparsity objective drives computation reduction while self-distillation preserves capability along the sparse execution path.

Table 4: Learning-method ablations on Qwen3.

Design	$\bar{K}$	Reasoning & Knowledge					Code		Instruction		Avg.	ΔAvg.
		MATH	AIME24	GPQA-D	MMLU-R	GSM8K	Eval+	MBPP+	IFEval	IFBench		
BEAM	4.03	93.60	76.25	60.10	89.29	94.31	90.24	78.57	83.36	40.33	78.45	–
w/o layer-wise KD	4.00	94.60	73.33	61.62	89.22	94.31	87.20	79.37	83.92	40.33	78.21	-0.3%
Soft-mask training	4.00	93.60	70.42	55.56	87.88	95.22	90.85	77.51	80.96	40.67	76.96	-1.9%
w/o sparsity objective	7.83	94.00	81.67	63.64	89.80	95.07	87.20	79.10	84.84	44.67	80.00	$+2.0\%$

Having established quality preservation and isolated the relevant design choices, we next ask whether the saved expert activations translate into wall-clock gains.

5.4 Efficiency Evaluation

Dynamic MoE kernels. To translate irregular masks into operator savings, our dynamic dispatch removes masked token–expert pairs before expert execution while reusing vLLM's optimized expert GEMMs (Figure 5). Across invocation sizes, BEAM approaches native Top-4 expert latency despite retaining a variable budget for each token.

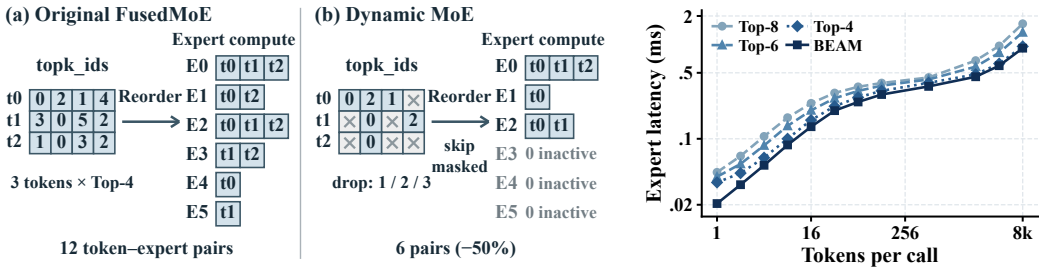


Figure 5: Dynamic dispatch removes masked assignments and approaches fixed Top-4 expert latency.

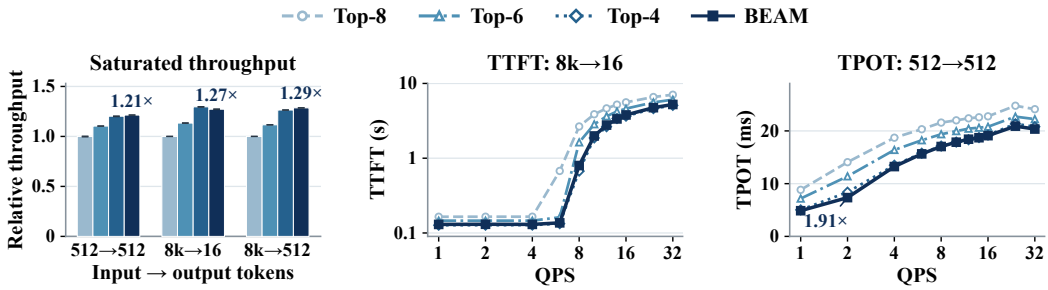


Figure 6: End-to-end throughput and latency under serving workloads.

End-to-end acceleration. Figure 6 evaluates whether these operator savings persist under end-to-end serving workloads. With CUDA Graph and saturated request queues, BEAM improves throughput by up to 1.29× over native Top-8 while remaining close to fixed Top-4 execution. Across request-rate sweeps, the reduced expert workload yields up to 1.91× TPOT speedup over Top-8.

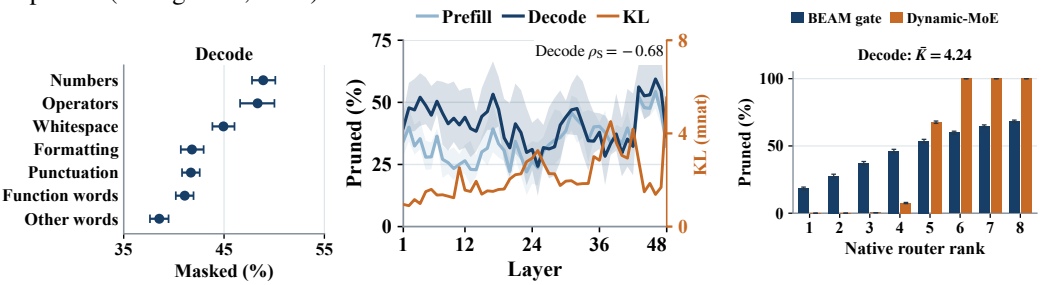
Training cost. Routing adaptation remains inexpensive because the pretrained backbone is frozen. The estimated SFT and OPSD stages each complete within a few hours on a single eight-GPU node (Table 5); detailed settings are provided in Appendix A.

Table 5: Training cost by stage.

Stage	GPUs	Time (h)
Qwen3 (SFT)	8×H800	4.50 (2 epochs)
GLM (SFT)	8×H800	5.11 (2 epochs)
Qwen3 (OPSD)	8×H20	3.04 (32 steps)
GLM (OPSD)	8×H20	4.77 (32 steps)

6 Analysis of Learned Expert Allocation

We finally ask whether BEAM learns the allocation structure predicted by the intervention studies in Section 3, examining its preferences across tokens, layers, and router ranks on Qwen3 MT-Bench responses (Zheng et al., 2023).



(a) Token categories **(b) Layer allocation** **(c) Candidate rank**

Figure 7: BEAM learns structured token-layer allocation while departing from prefix-based pruning.

Token allocation. Numbers and operators are masked at approximately 49%, about 10 percentage points more often than ordinary words (Figure 7a). The within-response examples in Figure 8 confirm that this variation persists among tokens generated in the same local context.

Layer sensitivity. We compare learned pruning with the static intervention in Figure 2a. Across 48 layers, static output KL and learned decode-time pruning have Spearman correlation $\rho = -0.68$ (Figure 7b), so layers that are more sensitive to truncation retain more computation. This correspondence emerges without layer-specific budget targets and connects the independent intervention to the allocation learned under a shared budget.

Candidate rank. At a matched mean count on the same routing states, Dynamic-MoE's Top-p rule concentrates pruning at later ranks, whereas BEAM's pre-safeguard pruning increases gradually from 18.3% at rank 1 to 68.3% at rank 8 (Figure 7c). BEAM can therefore remove earlier candidates while retaining later ones, consistent with the finding in Section 3.2 that native rank is a prior rather than a fixed deletion order. Appendix C.3 details the protocol and Appendix C.3 examines safeguard effects.

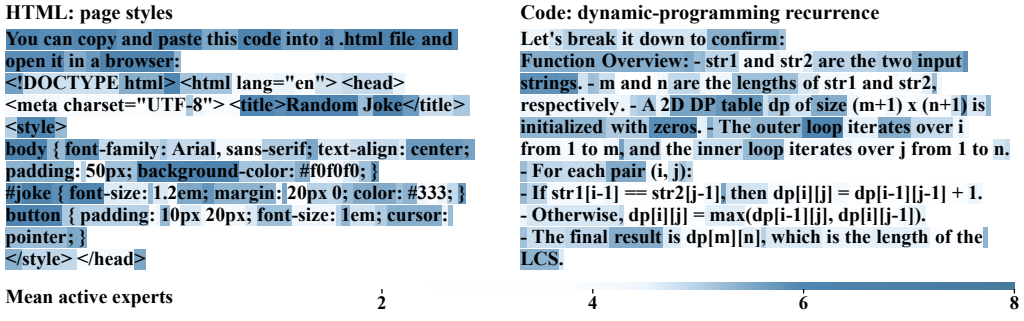


Figure 8: Within-response allocation. Darker backgrounds indicate more active experts.

7 Related Work

Training-free expert reduction. HC-SMoE merges experts through output-based hierarchical clustering (Chen et al., 2025); NAEE combines reconstruction-based pruning with routing-score-based skipping (Lu et al., 2024). MoDES uses layer importance and modality-specific thresholds (Huang et al., 2026), while SERE re-routes assignments using expert similarity within a decoding batch (Wu et al., 2026). These methods compress the expert pool or prescribe which activations to skip or redirect. BEAM instead preserves the expert bank and learns retention from sparse execution.

Adaptive activation budgets. Dynamic-MoE selects a probability-ranked prefix using Top-p (Huang et al., 2024). Ada-K learns expert counts with the original model frozen (Yue et al., 2024), while GeMoE trains an entropy-guided count predictor and router with experts fixed (Cai et al., 2026). Their selections remain prefixes of the frozen or adapted ranking; BEAM learns candidate-wise retention, allowing non-prefix subsets.

Learned sparse architectures. AdaMoE adds null experts (Zeng et al., 2024), and MoE++ combines FFN, zero, copy, and constant-mixture experts (Jin et al., 2024). DynMoE learns expert thresholds and adjusts the expert pool (Guo et al., 2025); ReMoE replaces Top-K selection with ReLU gates (Wang et al., 2025). ZEDA introduces zero-output experts and uses two-stage self-distillation (Lv et al., 2026). These approaches change the expert choices or gating function. BEAM pairs binary activation masks with sparse-path self-distillation, learning retention while keeping all pretrained parameters frozen.

8 Conclusion

Fixed Top-K routing leaves sparsity potential in pretrained expert banks. BEAM learns context-dependent retention through sparsity-guided self-distillation, preserving pretrained weights while reducing expert activation. Its learned allocation reflects pruning sensitivity, and dynamic MoE kernels translate skipped activations into practical inference gains. Experiments span two backbones, with serving results on Qwen3 (Appendix D).

Reproducibility Statement

Section 4 specifies the activation policy and frozen parameter scope. Appendix A details the training objectives and optimization recipe, and Appendix B records task sizes and scoring conventions. Appendix D specifies the operator and end-to-end inference measurements.

AI Use Statement

Generative AI tools, including OpenAI Codex, assisted with refining the research narrative and experimental comparisons, interpreting existing experiment records, literature organization, English drafting and translation, and LaTeX preparation. The pretrained model also generated the offline training trajectories described in Section 4.2. Numerical measurements were taken from the experiment records. The authors take responsibility for the manuscript and its use of AI assistance.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, et al. On-policy distillation of language models: Learning from self-generated mistakes. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 21246–21263, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/5be69a584901a26c521c2b51e40a4c20-Paper-Conference.pdf.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation, 2013. URL https://arxiv.org/abs/1308.3432.

Akhiad Bercovich, Itay Levy, Izik Golan, et al. Llama-nemotron: Efficient reasoning models, 2025. URL https://arxiv.org/abs/2505.00949.

Chaoxiang Cai, Minghe Weng, Jie Li, et al. Gemoe: Gating entropy is all you need for uncertainty-aware adaptive routing in moe-based large vision-language models, 2026. URL https://arxiv.org/abs/2606.26287.

I-Chun Chen, Hsu-Shen Liu, Wei-Fang Sun, Chen-Hao Chao, Yen-Chang Hsu, and Chun-Yi Lee. Retraining-free merging of sparse MoE via hierarchical clustering. In Aarti Singh, Maryam Fazel, Daniel Hsu, et al. (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8594–8620. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/chen25aq.html.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, et al. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022. URL http://jmlr.org/papers/v23/21-0998.html.

Trevor Gale, Deepak Narayanan, Cliff Young, and Matei Zaharia. Megablocks: Efficient sparse training with mixture-of-experts. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 288–304. Curan, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/file/5a54f79333768effe7e8927bcccffe40-Paper-mlsys2023.pdf.

Aryo Pradipta Gema, Joshua Ong Jun Leang, Giwon Hong, et al. Are we done with MMLU? In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5069–5096, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.262. URL https://aclanthology.org/2025.naacl-long.262/.

GLM Team, Aohan Zeng, Xin Lv, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models, 2025. URL https://arxiv.org/abs/2508.06471.

Yongxin Guo, Zhenglin Cheng, Xiaoying Tang, Zhaopeng Tu, and Tao Lin. Dynamic mixture of experts: An auto-tuning approach for efficient transformer models. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 79643–79672, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c63ff34c163bca3055d39490f6d1aaf7-Paper-Conference.pdf.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

Quzhe Huang, Zhenwei An, Nan Zhuang, et al. Harder task needs more experts: Dynamic routing in MoE models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12883–12895, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.696. URL https://aclanthology.org/2024.acl-long.696/.

Yushi Huang, Zining Wang, Zhihang Yuan, et al. Modes: Accelerating mixture-of-experts multimodal large language models via dynamic expert skipping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 30205–30215, June 2026.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng Yan. Moe++: Accelerating mixture-of-experts methods with zero-computation experts. *arXiv preprint arXiv:2410.07348*, 2024.

Zewen Jin, Shen Fu, Chengjie Tang, et al. Smidt: High-performance inference framework for moe models with dynamic top-k routing. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(27):22444–22453, March 2026. ISSN 2159-5399. doi: 10.1609/aaai.v40i27.39403. URL http://dx.doi.org/10.1609/aaai.v40i27.39403.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6980.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, et al. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, et al. Let's verify step by step. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/aca97732e30bcf1303bc22ac3924fd16-Paper-Conference.pdf.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and LINGMING ZHANG. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 21558–21572. Curran Associates, Inc., 2023. doi: 10.52202/075280-0943. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/43e9d647ccd3e4b7b5baab53f0368686-Paper-Conference.pdf.

Zihan Liu, Zhuolin Yang, Yang Chen, et al. Acereason-nemotron 1.1: Advancing math and code reasoning through sft and rl synergy. *arXiv preprint arXiv:2506.13284*, 2025.

Xudong Lu, Qi Liu, Yuhui Xu, et al. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6159–6172, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.334. URL https://aclanthology.org/2024.acl-long.334/.

Xingtai Lv, Li Sheng, Kaiyan Zhang, et al. Post-trained moe can skip half experts via self-distillation, 2026. URL https://arxiv.org/abs/2605.18643.

Alexandre Muzio, Alex Sun, and Churan He. Seer-moe: Sparse expert efficiency through regularization for mixture-of-experts, 2024. URL https://arxiv.org/abs/2404.05089.

Valentina Pyatkin, Saumya Malik, Victoria Graf, et al. Generalizing verifiable instruction following. In D. Belgrave, C. Zhang, H. Lin, et al. (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference. Curran Associates, Inc., 2025. doi: 10.52202/085713-1645. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/46499a0622ecf568b72d17b61e45dbd5-Paper-Datasets_and_Benchmarks_Track.pdf.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

David Rein, Betty Li Hou, Asa Cooper Stickland, et al. GPQA: A graduate-level google-proof q&a benchmark. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Ti67584b98.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets, 2015. URL https://arxiv.org/abs/1412.6550.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, et al. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, 2017. URL https://arxiv.org/abs/1701.06538.

Ziteng Wang, Jun Zhu, and Jianfei Chen. Remoe: Fully differentiable mixture-of-experts with relu routing. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 59486–59507, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/94dc604e115237a7f4a758b3146cd976-Paper-Conference.pdf.

Juntong Wu, Jialiang Cheng, Fuyu Lv, Ou Dan, and Li Yuan. Sere: Similarity-based expert re-routing for efficient batch decoding in moe models, 2026. URL https://arxiv.org/abs/2602.07616.

Tongtian Yue, Longteng Guo, Jie Cheng, Xuange Gao, and Jing Liu. Ada-k routing: Boosting the efficiency of moe-based llms, 2024. URL https://arxiv.org/abs/2410.10456.

Zihao Zeng, Yibo Miao, Hongcheng Gao, Hao Zhang, and Zhijie Deng. AdaMoE: Token-adaptive routing with null experts for mixture-of-experts language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 6223–6235, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.361. URL https://aclanthology.org/2024.findings-emnlp.361/.

Liujie Zhang, Benzhe Ning, Rui Yang, et al. Relax: An asynchronous reinforcement learning engine for omni-modal post-training at scale, 2026. URL https://arxiv.org/abs/2604.11554.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 46595–46623. Curran Associates, Inc., 2023. doi: 10.52202/075280-2020. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, et al. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A Model and Training Details

A.1 Base BEAM Adaptation

Only the layer-wise mask projections and biases are trainable; all pretrained model parameters remain frozen.

Table 6: Main Qwen3 BEAM adaptation setting.

Component	Setting
Backbone / native routing	Qwen3-30B-A3B / Top-8
Routed layers / experts / hidden width	48 / 128 / 2,048
Trainable parameters	12,589,056 mask parameters
Gate input / initialization	normalized pre-MoE hidden state / all candidates retained
Training data	53,969 native-model trajectories, two passes
Optimizer / learning rate / global batch	Adam / 10^{-5} / 16 sequences
Target count / budget magnitude	$K^\star = 4$ / $\beta = 0.1$
Distillation weights	layer-wise path 1, output KL 1, token CE 0
Maximum sequence length / precision	32,768 / BF16 model, FP32 mask projection
Hardware / framework	8×H800 / Relax

- Prompts are drawn from AceReason-1.1-SFT and Llama-Nemotron-Post-Training-Dataset; responses come from the frozen native model and introduce no external labels.
- Prompt and response tokens contribute to training, while packing padding is excluded.
- GLM-4.7-Flash uses the same mask-only recipe with native routed $K = 4$ and targets $K = 2$ or $K = 3$. Its shared expert is unchanged.

A.2 Exact Distillation and Budget Losses

For valid token positions $\mathcal{V}$, teacher and student hidden states are aligned with

$$D(\mathbf{H}_\ell^{\mathrm{S}}, \mathbf{H}_\ell^{\mathrm{T}}) = \frac{1}{|\mathcal{V}|} \sum_{t \in \mathcal{V}} \frac{\|\mathbf{h}_{\ell,t}^{\mathrm{S}} - \mathbf{h}_{\ell,t}^{\mathrm{T}}\|_2^2/d}{\max\{\|\mathbf{h}_{\ell,t}^{\mathrm{T}}\|_2^2/d,\, \bar{e}_\ell,\, 10^{-6}\}}, \quad \bar{e}_\ell = \frac{1}{|\mathcal{V}|} \sum_{t \in \mathcal{V}} \|\mathbf{h}_{\ell,t}^{\mathrm{T}}\|_2^2/d. \tag{10}$$

For output distillation, J_t contains the teacher's Top-32 vocabulary entries and all remaining probability mass is merged into one tail bin:

$$\mathcal{L}_{\mathrm{out}} = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \sum_{j \in J_t \cup \{\mathrm{tail}\}} q_{t,j}^{\mathrm{T}} \log \frac{q_{t,j}^{\mathrm{T}}}{q_{t,j}^{\mathrm{S}}}. \tag{11}$$

The normalized usage and signed budget price are

$$U_n = \frac{1}{K_0 |\mathcal{I}_n|} \sum_{i \in \mathcal{I}_n} \mathbf{1}_E^\top \mathbf{m}_i^{\mathrm{ST}}, \qquad \lambda_{n+1} = \beta\, \mathrm{sign}(\bar{K}_n - K^\star), \qquad \lambda_0 = 0. \tag{12}$$

Teacher targets are detached. The student propagates its own sparse states, and gradients update only the mask gates through the STE.

A.3 On-Policy Self-Distillation

Qwen3 OPSD starts from the completed hidden-input BEAM checkpoint and updates only mask weights and biases with fresh Adam states.

- **Prompts.** A fixed pool of 512 questions is balanced across four nominal sources (math, science, instruction, and code). Each update samples eight prompts; the 16-update run therefore uses 128 on-policy draws.

- **Rollouts.** The current sparse student generates one response per prompt with temperature 0.6, top-p 0.95, top-k 20, and a 32K response limit. The frozen native model evaluates the same generated prefixes.
- **Objective.** OPSD uses response-token forward KL over the teacher Top-32 entries plus a merged tail bin. It does not use layer-wise hidden-state KD or token CE:

$$\mathcal{L}_{\mathrm{OPSD}} = \frac{1}{|\mathcal{T}_{\mathrm{resp}}|} \sum_{t \in \mathcal{T}_{\mathrm{resp}}} \sum_{j \in J_t \cup \{\mathrm{tail}\}} q_{t,j}^{\mathrm{T}} \log \frac{q_{t,j}^{\mathrm{T}}}{q_{t,j}^{\mathrm{S}}}, \qquad (13)$$
$$\mathcal{L} = \mathcal{L}_{\mathrm{OPSD}} + \lambda_n U_n, \qquad \lambda_n = 0.02 \ \mathrm{sign}(\bar{K}_{n-1} - 4).$$

Table 7: Qwen3 OPSD checkpoints. Screen results use 50 examples per non-AIME task and full AIME24@8. Only step 4 is evaluated on the complete nine-task suite.

Checkpoint	$\bar{K}_{\mathrm{train}}$	MATH	AIME24	GPQA-D	MMLU-R	GSM8K	Eval+	MBPP+	IFEval	IFBench
Step 2 (screen)	3.95	92.00	77.92	60.00	100.00	100.00	92.00	90.00	88.00	54.00
Step 4 (screen)	3.86	92.00	79.17	58.00	98.00	100.00	92.00	94.00	88.00	50.00
Step 8 (screen)	4.25	90.00	76.25	64.00	98.00	100.00	92.00	90.00	86.00	66.00
Step 16 (screen)	3.99	92.00	75.00	64.00	98.00	100.00	90.00	86.00	86.00	56.00
Step 4 (full)	3.86	94.40	79.17	68.18	88.87	94.01	90.24	76.98	84.10	39.00

The screen is not a monotonic training curve: checkpoints 2, 4, 8, and 16 trade off different tasks. Step 4 was selected after its AIME24@8 screen and then evaluated on the full suite. Its reported $\bar{K}_{\mathrm{train}}$ is a pre-update training-batch statistic, not an inference-time count. The 16-update run uses 8×H20 GPUs and averages approximately 5.76 minutes per update.

A.4 Activation Safeguard

- The reported Qwen checkpoint executes the union of threshold-passing candidates and its original two-candidate router-weight safeguard. Surviving mixture weights are not renormalized.
- A rank-agnostic alternative fills an undersized selection using the largest final mask scores:

$$S_{\mathrm{fb}} = \begin{cases} S, & |S| \geq k, \\ \mathrm{TopK}_k(\mathbf{s}|_C), & |S| < k, \end{cases} \qquad S = \{e \in C : s_e \geq 1/2\}. \qquad (14)$$

This fallback is analyzed descriptively with the checkpoint fixed; the reported quality and speed numbers use the original checkpoint rule.

B Evaluation Data and Scoring

Table 8: Main evaluation datasets and metrics.

Benchmark	Problems	Metric
MATH-500 (Lightman et al., 2024)	500	Answer accuracy
AIME24@8	30	Average accuracy over eight samples per problem
GPQA-D (Rein et al., 2024)	198	Multiple-choice accuracy
MMLU-R (Gema et al., 2025)	5,519	Accuracy
GSM8K (Cobbe et al., 2021)	1,319	Answer accuracy
Eval+ (Liu et al., 2023)	164	HumanEval+ pass@1
MBPP+ (Liu et al., 2023)	378	Pass@1
IFEval (Zhou et al., 2023)	541	Strict prompt-level accuracy
IFBench (Pyatkin et al., 2025)	300	Loose prompt-level accuracy

- Accuracy evaluation uses temperature 0.6, top-p 0.95, top-k 20, a 32K output limit, disabled prefix caching, and CUDA Graph execution.

- Retention $\mathcal{R}$ is the equal-weight mean of task-wise ratios to the matched native model. AIME24 reports Avg@8 rather than pass@8.
- Router-only and Full Finetune use fixed reduced budgets. Dynamic-MoE, Ada-K, and ZEDA use the configurations reported in Table 2. The Qwen OPSD row is the step-4 checkpoint in Table 7.
- In Table 4, the soft-mask alternative is evaluated by retaining the four largest final mask scores at inference.

C Diagnostics and Learned Allocation

C.1 Intervention Summary

- **Rank removal.** Each original Top-8 candidate is removed independently with the remaining IDs and coefficients fixed. The diagnostic measures next-token KL without survivor renormalization.
- **Layer sensitivity.** Whole-layer probes reduce all tokens at one layer and measure final-output KL after propagation through the remaining model.
- **Token sensitivity.** Token probes prune either one token across layers or one token-layer site, revealing variation across token positions and depth.

C.2 Token and Token-Layer Allocation

The following views use the same saved coding response and the same active-expert scale. The upper panel averages each token over 48 MoE layers; the lower panel resolves the executed count at every token-layer site.

(a) Token-level allocation (mean across 48 layers)

```
You can copy and paste this code into a .html file and open it in a browser:
<!DOCTYPE html> <html lang="en"> <head>
<meta charset="UTF-8"> <title>Random Joke</title> <style>
body { font-family: Arial, sans-serif; text-align: center; padding: 50px; background-color: #f0f0f0; }
#joke { font-size: 1.2em; margin: 20px 0; color: #333; }
button { padding: 10px 20px; font-size: 1em; cursor: pointer; }
</style> </head>
```

(b) Token-layer allocation for the same excerpt

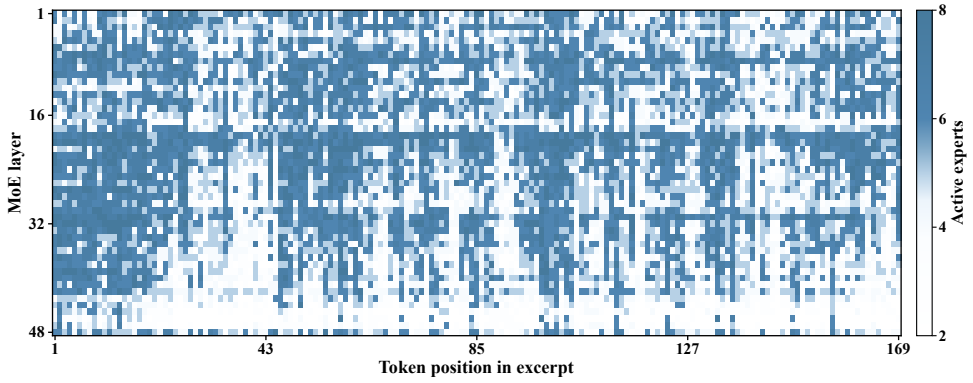


Figure 9: Token- and token-layer expert allocation for the same response excerpt. Darker cells indicate more active experts. Statistics are obtained by causal replay of the reported hidden-input checkpoint at threshold 0.5.

C.3 Learned Allocation Protocol

- The main allocation analysis replays 80 first-turn MT-Bench responses (eight task types, ten responses each) and records token-, layer-, and rank-level pruning.
- Token categories are fixed lexical groups. Reported rates average positions and layers within each response before averaging responses equally.
- The layer comparison uses all 48 layers. Static output KL and decode pruning have Spearman correlation $\rho = -0.68$; prefill correlation is near zero.
- The rank comparison matches BEAM and Dynamic-MoE at mean $\bar{K} = 4.24$ on the same saved routing states. BEAM uses pre-safeguard gate decisions; Dynamic-MoE retains a Top-p prefix.
- The alternative gate-score safeguard is replayed on the same fixed texts. It is a diagnostic policy variant and does not inherit the reported quality or speed results.

D Inference Implementation and Measurement

D.1 Sparse Execution

The dynamic operator evaluates the mask before expert dispatch, removes masked token–expert pairs, and reuses the native expert GEMMs. Surviving mixture coefficients and all resident expert weights remain unchanged.

D.2 Reported Measurement Settings

- **Expert operator (Figure 5).** Qwen3-30B-A3B, one H800, BF16, tensor parallelism one, vLLM 0.10.2, and CUDA Graph timing. The curve reports expert execution only and excludes router and mask projections.
- **Saturated throughput (Figure 6, left).** One H800, BF16, tensor parallelism one, vLLM 0.29, and CUDA Graph execution. Workloads are $512 \to 512$, $8{,}192 \to 16$, and $8{,}192 \to 512$ tokens. All requests are available at launch and throughput counts real output tokens only. Graph-mode checks give $\bar{K} = 3.96$, 4.01, and 4.01.
- **Request-rate sweeps (Figure 6, middle/right).** TTFT uses $8{,}192 \to 16$ and TPOT uses $512 \to 512$ over QPS 1–32, with 96 requests per point. The peak TPOT ratio occurs at QPS 2.
- **Common settings.** Prefix caching is disabled, output lengths are fixed with EOS ignored, and methods use matched requests on the same GPU. Retention thresholds are fixed within each workload. Serving and quality evaluation use separately calibrated operating points.

The low-QPS latency sweeps supplement the saturated throughput experiment; they do not estimate steady-state serving capacity. HTTP and network overhead are excluded.