ExpertNavigator: Functionally Coherent Expert Grouping and Pairwise-Ranked Routing for High-Fidelity Dense-to-MoE Conversion

Xiaohan Qin[1,2,*], **Cancheng Zhang**[1,*], **Xiaoxing Wang**[1], **Xiangdong Zhang**[1], **Yebin Yang**[1], **Ning Liao**[1], **Junchi Yan**[1,†]
[1]School of AI, Shanghai Jiao Tong University, [2]Shanghai Innovation Institute

Abstract

Dense large language models (LLMs) exhibit implicit sparsity at inference time: for a given token, only a small subset of FFN neurons contributes dominantly. This observation motivates *Dense-to-MoE conversion*, which partitions FFN neurons into experts and employs a lightweight router to sparsely activate experts for efficiency. However, existing conversion methods often incur non-trivial quality degradation under a fixed activation budget, largely due to suboptimal expert construction and unstable router learning in fine-grained settings. Motivated by this, we present **ExpertNavigator**, which improves Dense-to-MoE conversion with a more functionally coherent expert grouping method and a more stable and effective pairwise-ranked router training strategy, leading to markedly better quality under the same activation budget. Moreover, we further introduce an adaptive layer-wise sparsity allocation strategy to better utilize a global activation budget, and apply STE-based joint training to enhance model–router compatibility under hard routing. Experiments across various benchmarks show that ExpertNavigator consistently outperforms prior Dense-to-MoE approaches under different sparsity budgets, retaining up to **97.8%** of the dense model's average score while activating only 60% of FFN parameters, substantially narrowing the gap to dense models and delivering practical efficiency gains. The project page is available at https://github.com/jianke0604/ExpertNavigator.

1 Introduction

As the parameter scale continues to grow, the capabilities of large language models (LLMs) have improved substantially [31, 49, 58], yet inference-time computation and latency remain major bottlenecks for real-world deployment. Recent studies [6, 32] have shown that the forward computation of dense models often exhibits pronounced implicit sparsity: for a given input token, only a small subset of channels/neurons contributes the dominant increment, while a large fraction of parameters stays effectively dormant. Consequently, exploiting such implicit sparsity to accelerate inference without sacrificing model quality has become a promising research direction.

Existing approaches typically fall into two categories: *structured pruning* [33, 38] and *dynamic sparse activation* [40, 65]. The former permanently removes a portion of model parameters, enabling deterministic speedups, but it often degrades representational capacity and disrupts functional structures, leading to noticeable quality loss [18]. The latter preserves the full parameter capacity while activating only a subset of computation paths via input-dependent selection, aiming to offer greater flexibility and higher potential capability. Such sparse activation is typically designed around the

*Equal contribution. †Corresponding author: yanjunchi@sjtu.edu.cn.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

FFN component of dense LLMs, which commonly accounts for a dominant portion of FLOPs, and is therefore often referred to as *Dense-to-MoE conversion*.

Despite the promising inference speedups enabled by sparse activation, models produced by recent Dense-to-MoE conversion methods still exhibit a non-trivial performance gap compared to their original dense counterparts, especially when constrained to a limited activation budget. We posit that this gap is largely driven by two key challenges:

(1) How to obtain well-structured and behaviorally aligned expert groups? In Dense-to-MoE conversion, experts do not arise from end-to-end co-training; instead, they are constructed by partitioning the dense FFN. Consequently, expert grouping becomes a non-trivial design challenge, as its quality largely influences the performance of the resulting sparse model. However, most existing grouping strategies either rely on parameter-space similarity (e.g., weight-based clustering [47]) or exploit activation statistics from only the gating pathway (e.g., gate-only co-activation [63]). Such signals can be insufficient in modern FFNs with SwiGLU-style architectures, where a neuron's effective contribution is jointly governed by multiple branches (i.e., W_g, W_{up}, and W_{down}); focusing on a single pathway therefore yields an incomplete characterization of functional structure. More critically, suboptimal grouping can induce forced co-activation or co-suppression at inference time: neurons that should be selectively activated in the dense model become artificially coupled within the same expert, thereby disrupting the knowledge organization and causing substantial quality degradation.

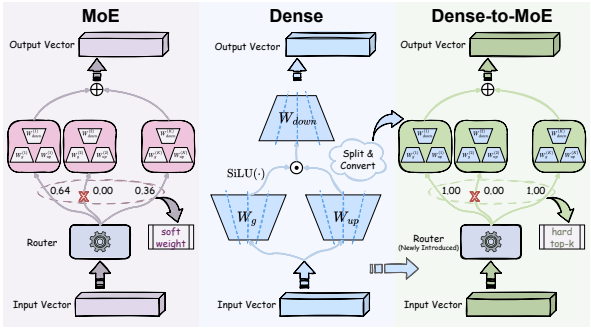


Figure 1: Comparison between MoE and Dense-to-MoE conversion. The latter slices FFN weights from a dense LLM into experts and adds a lightweight router for hard top-k inference.

(2) How to enable stable and effective router training? As shown in Fig. 1, unlike conventional MoE, Dense-to-MoE typically follows a split-then-train pipeline, where experts are fixed first and the router is trained afterwards. Under this setting, naively adopting the standard softmax-based routing objective from the MoE paradigm can lead to severe optimization instability and even catastrophic performance collapse [65]. Some existing router-training schemes attempt to regress the absolute magnitude of expert activation strength or output norms. However, under fine-grained expert partitioning, token-wise activations vary widely across contexts, the number of experts is large, and expert outputs can differ substantially in scale, making such absolute targets difficult to fit. As a result, the router struggles to train stably and learn reliable selection policies.

Motivated by these observations, we propose **ExpertNavigator**, a systematic method for high-fidelity Dense-to-MoE conversion with targeted designs across four aspects: *expert grouping*, *router training*, *layer-wise sparsity allocation*, and *model–router adaptation*. First, for expert construction, we argue that parameter-space similarity or gate-only activation statistics are insufficient to capture the true functionality of a neuron group (spanning W_g, W_{up}, and W_{down}). We therefore propose a function-consistency-based grouping strategy to better preserve the implicit functional structure of dense FFNs (Sec. 3.1). Second, for router training, we emphasize that the routing goal in Dense-to-MoE is inherently a discrete top-k selection problem (Fig. 1): the router only needs to decide whether an expert is selected, without regressing absolute response scales or producing continuous expert weights. Accordingly, we introduce a pairwise rank distillation objective that learns the relative ordering of expert contributions to improve training stability and generalization (Sec. 3.2).

Furthermore, we observe substantial cross-layer heterogeneity in sensitivity to activation budgets, implying that a uniform per-layer sparsity strategy is suboptimal under a fixed global budget. We thus propose an adaptive layer-wise sparsity allocation mechanism that distributes per-layer activation ratios according to model characteristics under a total compute budget (Sec. 3.3). Finally, to further enhance model–router compatibility under strict hard top-k routing across different sparsity settings, we adopt a joint training strategy based on the straight-through estimator (STE), improving performance under varying sparsity levels (Sec. 3.4). Our contribution can be summarized as follows:

1). Functionally coherent expert grouping method. We propose an expert construction strategy that more closely reflects the actual computational behavior of neurons, addressing the limitations of prior grouping approaches based on parameter similarity or single-path activation statistics.

2). Pairwise-ranked router training strategy. Motivated by the discrete selection nature of Dense-to-MoE routing, we propose a pairwise ranking distillation objective that replaces absolute-scale fitting with relative-order supervision, improving the stability and quality of router training.

3). Adaptive layer-wise sparsity allocation. We introduce a layer-wise sparsity allocation strategy that adaptively distributes activation budgets across layers under a fixed overall budget, further improving the quality–efficiency trade-off over a uniform per-layer sparsity strategy.

4). Superior performance across various benchmarks. We further apply the STE-based model–router adaptation on the final hard top-k model to obtain additional gains under diverse sparsity levels. Experiments across different model families show that ExpertNavigator consistently outperforms prior Dense-to-MoE methods across diverse sparsity settings, retaining up to **97.8%** of the average benchmark score while activating only **60%** of FFN parameters.

2 Related Work

Structured pruning. A prevalent approach to leverage sparsity is static structured pruning [13, 33, 36, 46, 57], which identifies and permanently removes structured parameters/submodules via importance estimation, yielding a smaller model and deterministic inference speedups. Representative approaches can be categorized as weight magnitude [5, 15], loss curvature [16, 27], and structural redundancy [2, 30, 38]. However, due to the irreversible capacity reduction and potential disruption of learned functional structure, structured pruning often incurs non-trivial quality degradation, especially under high-fidelity requirements [18].

Dynamic sparse activation. Another stream of work is dynamic sparse activation, which preserves the full parameter capacity but reduces inference cost by activating only a subset of computation paths in an input-dependent manner [9, 32]. Since the FFN typically dominates inference FLOPs [53] and exhibits pronounced contextual sparsity [32], recent studies increasingly focus on sparsifying FFN computations. Inspired by the MoE paradigm [45], these methods usually partition dense FFN neurons into expert groups and introduce an additional lightweight router to perform top-k selective activation, commonly referred to as *Dense-to-MoE conversion* [40, 47, 63, 65]. We summarize the detailed pipelines of several recent Dense-to-MoE methods in Appendix D. Unlike *upcycling* approaches that construct larger MoE models by duplicating or expanding dense weights [20, 29, 39], this conversion primarily aims to exploit the dense model's implicit sparsity for efficient inference rather than increasing model capacity. However, existing methods still face the two key challenges in Sec. 1, and often incur non-trivial quality degradation under a fixed activation budget. In Sec. 3, we introduce ExpertNavigator and show how it addresses these challenges in a targeted manner.

3 Methods

3.1 Functionally Coherent Expert Grouping

Expert grouping is a shared first step in most Dense-to-MoE conversion methods. We consider a modern SwiGLU-style FFN, whose parameters consist of $W_g \in \mathbb{R}^{d\times d_h}$, $W_{\text{up}} \in \mathbb{R}^{d\times d_h}$, and $W_{\text{down}} \in \mathbb{R}^{d_h\times d}$. Given an input token embedding $x \in \mathbb{R}^d$, the FFN computation can be written as

$$\text{FFN}(x) \;=\; h\,W_{\text{down}},\; h = \text{SiLU}(xW_g) \;\odot\; (xW_{\text{up}}). \tag{1}$$

Expert grouping specifies (i) the number of experts K, and (ii) a partition operator $\pi(\cdot)$ that slices the FFN weights into K disjoint groups. Concretely, we partition

$$\pi(W_g) = \{W_g^{(1)}, \ldots, W_g^{(K)}\},\; W_g^{(i)} \in \mathbb{R}^{d\times(d_h/K)}, \tag{2}$$

and analogously for W_{up} and W_{down}. We define the i-th expert E_i by its associated weights $\{W_g^{(i)}, W_{\text{up}}^{(i)}, W_{\text{down}}^{(i)}\}$, and its computation as

$$E_i(x) \;=\; \Big(\text{SiLU}(xW_g^{(i)}) \odot (xW_{\text{up}}^{(i)})\Big)\, W_{\text{down}}^{(i)}. \tag{3}$$

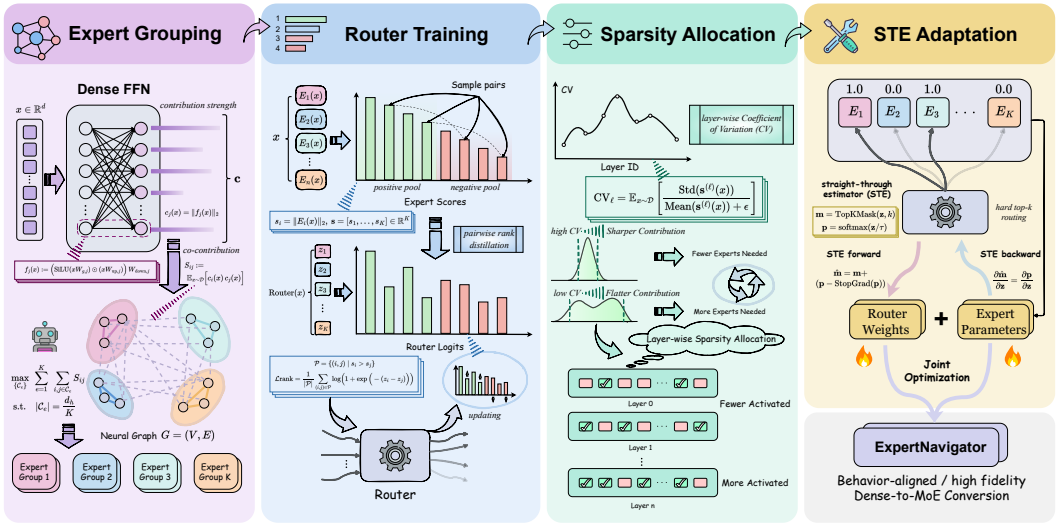


Figure 2: Overview of the proposed ExpertNavigator pipeline, consisting of four stages: expert grouping, router training, layer-wise sparsity allocation, and STE-based adaptation.

With this tensor-parallel style slicing, the original dense FFN can be recovered by summing all expert outputs: $\mathrm{FFN}(x) = \sum_{i=1}^{K} E_i(x)$. In Dense-to-MoE conversion, inference activates only a subset of experts for each token. Therefore, the partition operator π should ideally produce experts whose neurons exhibit similar input-response patterns within each expert, while maintaining sufficient separation across experts. This design is challenging in modern LLMs: the FFN hidden width d_h is typically very large (often on the order of 10^5 across model families), and neuron responses can vary substantially across inputs and contexts, making it non-trivial to construct expert groups that preserve the functional structure of the original dense FFN.

Prior methods typically design the partition operator π based solely on the gating matrix W_g, assuming it sufficiently distinguishes neuron activations. Two common choices are: (i) clustering W_g in parameter space with equal-size constraints (e.g., $\mathrm{KmeansConstrained}(W_g)$); and (ii) computing pairwise co-activation counts from gate activations $\mathrm{SiLU}(xW_g)$ over a dataset $\mathcal{D}$, and grouping neurons accordingly. While effective for earlier ReLU-based LLMs, these W_g-centric strategies are less suitable for modern SwiGLU FFNs: SiLU is a soft activation, and neuron contributions are jointly controlled by both the gate and up branches, making gate-only signals an incomplete proxy for functional behavior.

Based on these observations, we argue that expert grouping should jointly consider the functionality of the entire neuron tuple. Concretely, for neuron j with parameters $\{W_{g,j},\ W_{\mathrm{up},j},\ W_{\mathrm{down},j}\}$, its computation on an input embedding x can be written as

$$f_j(x) \ := \ \Big(\mathrm{SiLU}(xW_{g,j}) \odot (xW_{\mathrm{up},j})\Big)\, W_{\mathrm{down},j}. \tag{4}$$

We define the norm of $f_j(x)$ as the neuron's *contribution strength* to the token, $c_j(x) \ := \ \|f_j(x)\|_2$, which depends on W_g, W_{up}, and W_{down} simultaneously, and thus captures the neuron's actual role in the full FFN computation rather than relying on the gating branch alone. We further define the functional affinity between two neurons i and j as their *co-contribution* under the data distribution: $S_{ij} \ := \ \mathbb{E}_{x\sim\mathcal{D}}\big[c_i(x)\, c_j(x)\big]$. When two neurons tend to exhibit large contributions on many tokens, S_{ij} becomes large, indicating that they are likely to cooperate under similar contexts. This yields a weighted graph $G = (V, E)$ whose nodes are neurons and edge weights are given by S_{ij}. Unlike gate-based co-activation graphs [63] that add a binary $(0/1)$ count depending on whether two neurons are simultaneously activated by W_g, our S_{ij} explicitly preserves continuous contribution strengths.

Expert grouping can then be formulated as an equal-size graph partitioning problem: under the constraint that each expert contains exactly d_h/K neurons, we maximize the within-expert affinity (equivalently, minimize the cut),

$$\max_{\{\mathcal{C}_e\}} \sum_{e=1}^{K} \sum_{i,j\in\mathcal{C}_e} S_{ij} \quad \text{s.t.} \quad |\mathcal{C}_e| = \frac{d_h}{K}. \tag{5}$$

In practice, we find that a simple greedy equal-size clustering strategy [37] is already sufficient for solving this partitioning problem at the scale of modern FFNs, and we use it to obtain per-layer expert assignment indices.

3.2 Pairwise-ranked Router Training

Dense-to-MoE conversion augments each layer with a lightweight router to predict expert activation patterns. Recent studies typically introduce an additional router-training stage [32, 47], where all other parameters are frozen and the router is trained with extra supervision such that its predictions better match the grouped expert structure. As shown in Fig. 1, in Dense-to-MoE inference, routing is ultimately implemented as hard top-k expert selection: a binary mask activates the selected experts (with equal weights) while all others are suppressed. Therefore, we propose that the router primarily needs to learn the relative importance (ordering) among experts to determine which experts should be placed in the top-k, rather than directly matching hard-to-fit absolute output magnitudes.

Following [47] and consistent with the expert-grouping metric in Sec. 3.1, we use expert output norms as the teacher signal for ranking supervision. For an input $x \in \mathbb{R}^d$, we define the teacher score of expert i as

$$s_i = \|E_i(x)\|_2, \ \mathbf{s} = [s_1, \ldots, s_K] \in \mathbb{R}^K. \tag{6}$$

The router outputs logits $\mathbf{z} = \mathrm{Router}(x) \in \mathbb{R}^K$. Rather than regressing $\mathbf{z}$ to match the absolute scale of $\mathbf{s}$, we distill the *relative ordering* induced by $\mathbf{s}$ into $\mathbf{z}$.

Concretely, for any pair of experts (i, j), if the teacher indicates that expert i is more important than j (i.e., $s_i > s_j$), we form an ordered supervision pair $(i \succ j)$. Let $\mathcal{P} = \{(i, j) \mid s_i > s_j\}$ denote the set of sampled pairs for the current input. We train the router using a RankNet-style pairwise ranking loss:

$$\mathcal{L}_{\mathrm{rank}} = \frac{1}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \log\Big(1 + \exp\big(-(z_i - z_j)\big)\Big), \tag{7}$$

This objective directly aligns with hard top-k routing: selecting experts depends on their relative scores, and Eq. (7) enforces the correct pairwise ordering by increasing the margin between logits of preferred and less-preferred experts. Compared to regressing absolute expert activation magnitudes or output norms, the pairwise formulation is substantially less sensitive to global scale shifts and cross-expert scale heterogeneity. By supervising only the ordering—i.e., which experts should rank above others—the router can be trained more stably and generalize better to diverse inputs.

In practice, we sort the teacher scores $\mathbf{s}$ to form a positive pool and a negative pool, and sample 2048 pairs per input. Following [47, 63], we implement the router as a lightweight MLP, and find that router training converges quickly with only ~ 0.5B tokens. Additional sampling details are provided in Appendix B.1.

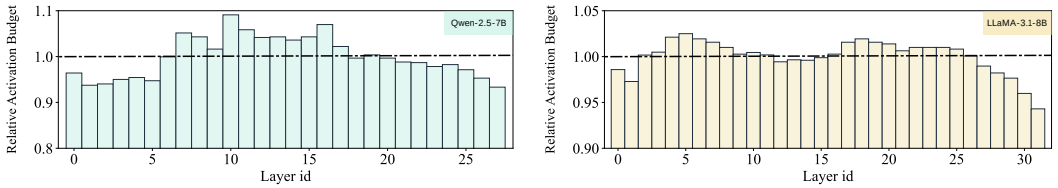
3.3 Adaptive Layer-wise Sparsity Allocation

During our experiments, we observe an intriguing phenomenon: router training exhibits noticeably different convergence behaviors across layers, in terms of both convergence speed and the final training loss. This suggests that expert specialization and activation patterns are not uniform throughout the network. In some layers, experts are strongly specialized: for a given token, only a small subset of experts attains high contribution strength, so activating only these dominant experts can already approximate the original dense FFN behavior. In other layers, expert contributions are more diffuse and cooperative, and activating more experts tends to yield better performance.

Given that Dense-to-MoE conversion typically introduces hundreds of experts per layer to better preserve the original activation patterns, we propose an *adaptive layer-wise sparsity allocation* strategy: under a fixed overall activation (or sparsity) budget, we assign different per-layer top-k budgets based on layer characteristics, aiming to outperform a uniform sparsity scheme that uses the same k for all layers.

Specifically, we use the coefficient of variation (CV), a standard statistic for measuring relative dispersion (scale-normalized variability), to characterize layer-wise expert-score distributions:

$$\mathrm{CV}_\ell = \mathbb{E}_{x\sim\mathcal{D}}\left[\frac{\mathrm{Std}(\mathbf{s}^{(\ell)}(x))}{\mathrm{Mean}(\mathbf{s}^{(\ell)}(x)) + \epsilon}\right], \tag{8}$$



(a) Qwen-2.5-7B (b) LLaMA-3.1-8B

Figure 3: Relative layer-wise activation budgets allocated by our adaptive sparsity scheme for Qwen-2.5-7B and LLaMA-3.1-8B.

where ℓ indexes layers and ϵ is a small constant for numerical stability. The CV reflects how concentrated or diffuse the expert scores $\mathbf{s}^{(\ell)}$ are within a layer. A larger CV_ℓ indicates a more peaked distribution, meaning that a small number of top experts covers most of the contribution and the layer can be sparsified more aggressively. Conversely, a smaller CV_ℓ suggests more uniformly distributed contributions across experts, and the layer should receive a larger activation budget.

Consistent with the data pool used for expert grouping, we use a small subset of high-quality pretraining data and compute the layer-wise CV statistics. We then convert $\{\mathrm{CV}_\ell\}$ into per-layer activation budgets by taking an inverse transform followed by a temperature-controlled softmax:

$$w_\ell = \frac{1}{\mathrm{CV}_\ell + \epsilon}, \qquad p_\ell = \frac{\exp(w_\ell/T)}{\sum_{i=1}^{L} \exp(w_i/T)}, \tag{9}$$

where T is typically set to a relatively large value to avoid overly large inter-layer disparities (e.g., $T = 20$). Fig. 3 visualizes the relative layer-wise activation budgets for Qwen-2.5-7B and LLaMA-3.1-8B. Despite belonging to different model families and having different depths, the CV-guided allocations exhibit a consistent trend: both shallow and deep layers are assigned lower activation budgets than uniform sparsity, whereas intermediate layers receive higher budgets. This pattern is consistent with prior findings [24, 51] that shallow layers (near the embedding) and deep layers (near the LM head) tend to have more predictable activations for token representations and can work well with fewer activated experts, while intermediate layers play a more critical role in semantic composition and moderately complex concept processing, thus benefiting from activating more experts. As expected, the ablation in Sec. 4.3 shows that introducing layer-wise sparsity allocation improves performance on both Qwen and LLaMA.

3.4 STE Joint Training & Overall Framework

Dense-to-MoE conversion performs inference with hard top-k routing, rather than the continuous mixture weights produced by softmax/sigmoid in pretrained MoE models. This discrete selection makes the routing operation non-differentiable, so naively continuing training with standard backpropagation is not straightforward. As a result, many prior conversion methods stop after the router-training stage [47, 63]. However, in Dense-to-MoE, experts are fixed by grouping while the router is trained separately, leaving a persistent model–router mismatch. Moreover, since the activated experts constitute only a subset of the original dense FFN, the MoE output can have a smaller scale than the dense output; such bias may accumulate with depth, leading to substantial degradation, especially under aggressive sparsity.

To address these issues, we introduce an additional full-parameter adaptation stage to improve model-router compatibility while preserving the potential for further post-conversion improvement. Since gradients cannot be propagated through the hard top-k selection directly, we adopt the straight-through estimator (STE) [3, 22] to provide an approximate gradient for the discrete routing decisions.

Concretely, for router logits $\mathbf{z} = \mathrm{Router}(x) \in \mathbb{R}^K$, we form a hard top-$k$ mask $\mathbf{m} = \mathrm{TopKMask}(\mathbf{z}, k) \in \{0,1\}^K$ and a temperature-scaled softmax distribution $\mathbf{p} = \mathrm{softmax}(\mathbf{z}/\tau)$, where τ is the STE temperature. We then use the STE gate

$$\hat{\mathbf{m}} = \mathbf{m} + (\mathbf{p} - \mathrm{StopGrad}(\mathbf{p})), \tag{10}$$

which satisfies the following forward/backward behaviors:

$$\textbf{Forward: } \hat{\mathbf{m}} = \mathbf{m} \text{ (strict hard top-}k\text{)}; \quad \textbf{Backward: } \frac{\partial \hat{\mathbf{m}}}{\partial \mathbf{z}} = \frac{\partial \mathbf{p}}{\partial \mathbf{z}}. \tag{11}$$

Table 1: Performance comparison of different Dense-to-MoE methods on two base models under 30% and 60% activation budgets. The best results are highlighted in **bold**. We also report the ratio of each method's average benchmark score to that of the original dense model.

Methods	Act.	MMLU	SciQ	PIQA	MathQA	ARC-C	ARC-E	HellaS.	NQ.	WinoG.	Avg.
Base Model: Qwen-2.5-7B											
Dense	100%	74.18	96.60	78.78	43.18	50.17	80.43	59.98	22.91	73.09	64.37 100%
CMoE	30%	26.41	73.40	56.80	21.71	18.86	34.13	28.48	0.14	52.17	34.68 53.9%
MoEfication	30%	23.51	63.20	60.94	21.88	25.34	42.30	32.94	0.25	48.86	35.47 55.1%
LTE	30%	27.87	86.50	66.59	23.25	28.33	57.28	37.94	1.14	56.99	42.88 66.6%
D2DMoE	30%	49.76	85.50	70.51	28.68	32.94	56.52	44.05	6.59	57.06	47.96 74.5%
ExpertNavigator	30%	**56.97**	**94.90**	**76.66**	**34.51**	**46.08**	**77.02**	**51.18**	**11.80**	**65.67**	**57.20** 88.9%
CMoE	60%	46.60	93.40	67.52	28.14	34.13	61.32	41.48	3.99	58.25	48.31 75.1%
MoEfication	60%	55.44	93.80	72.31	33.07	40.78	72.01	49.07	8.20	58.64	53.70 83.4%
LTE	60%	42.93	93.10	74.92	29.21	41.38	74.41	48.28	6.87	63.30	52.71 81.9%
D2DMoE	60%	66.39	94.70	77.04	39.53	44.28	75.25	55.69	16.01	63.14	59.11 91.8%
ExpertNavigator	60%	**69.47**	**96.00**	**79.16**	**42.51**	**50.60**	**81.10**	**56.24**	**20.03**	**71.67**	**62.98** 97.8%
Base Model: LLaMA-3.1-8B											
Dense	100%	65.08	96.20	80.03	39.53	51.45	81.57	60.01	27.70	73.56	63.90 100%
CMoE	30%	25.52	53.80	54.57	21.84	18.69	30.39	27.40	0.17	49.49	31.32 49.0%
MoEfication	30%	26.01	81.10	66.59	23.28	24.57	48.27	35.51	1.50	53.91	40.08 62.7%
LTE	30%	28.44	88.10	67.68	23.75	30.55	49.62	45.10	3.19	57.93	43.82 68.6%
D2DMoE	30%	26.74	80.20	66.65	25.26	26.96	53.45	37.80	1.25	55.17	41.50 64.9%
ExpertNavigator	30%	**38.33**	**92.80**	**73.07**	**29.55**	**36.18**	**70.08**	**47.38**	**7.29**	**58.64**	**50.37** 78.8%
CMoE	60%	32.02	92.90	69.04	28.44	32.00	60.94	44.27	5.37	60.69	47.30 74.0%
MoEfication	60%	50.44	94.00	77.09	32.43	39.68	70.37	51.02	10.58	67.72	54.81 85.8%
LTE	60%	33.17	93.20	74.81	27.64	39.59	71.59	49.56	5.21	63.61	50.93 79.7%
D2DMoE	60%	51.96	93.20	76.44	32.83	41.72	71.59	52.82	10.58	67.80	55.44 86.8%
ExpertNavigator	60%	**56.99**	**95.80**	**77.75**	**35.31**	**45.90**	**78.45**	**55.88**	**13.93**	**70.30**	**58.92** 92.2%

We apply $\hat{\mathbf{m}}$ to the selected experts, so the loss can jointly update the router and FFN parameters while keeping hard routing in the forward pass.

During this stage, we optimize all model parameters using the standard LM loss. To ensure a fair comparison with prior works, we perform adaptation with only a small amount of data ($<$ 1B tokens), which is orders of magnitude smaller than the $>$ 10T tokens used to pretrain the original dense model. Fig. 2 provides an overview of the proposed ExpertNavigator pipeline, and the overall algorithmic workflow is presented in Appendix E.

4 Experiments

4.1 Experimental Setup

Data Preparation. All data used in our experiments are drawn from the open-source Ultra-FineWeb pretraining corpus [52]. For expert clustering across all methods, we use a fixed subset of 1,000 samples, each with a sequence length of 2,048. Router training and STE-based model–router adaptation are each performed with approximately 0.5B tokens.

Baselines. We conduct dense-to-MoE conversions on three widely used open-source LLMs, Qwen-2.5-7B [43], LLaMA-3.1-8B [19], and larger Qwen-2.5-14B [43]. Following prior approaches [47, 65], we construct fine-grained experts, where each expert corresponds to a group of 16 neurons, inducing structured sparsity while largely preserving the original activation patterns; to assess the robustness of our conclusions to expert granularity, we additionally evaluate a coarser expert setting in Appendix B.5. Under this unified setup, we compare our method against several representative dense-to-MoE baselines, including MoEfication [63], D2DMoE [47], CMoE [40], and LTE [65]. Detailed descriptions of these baselines are provided in Appendix D.

Evaluation. We compare our method with baselines on a diverse suite of benchmarks, including MMLU [21], NQ-Open [28], SciQ [54], PIQA [4], MathQA [1], ARC-Challenge/Easy [10], HellaSwag [60], and WinoGrande [44]. Together, these benchmarks provide a comprehensive evaluation of factual knowledge, commonsense reasoning, mathematical problem solving, and multi-step logical inference. All evaluations are conducted using the standard `lm-eval-harness` [17] framework.

Table 2: Ablation study of the effectiveness of each component in ExpertNavigator. We report results on both Qwen-2.5-7B and LLaMA-3.1-8B at 60% expert activation.

Methods	MMLU	SciQ	PIQA	MathQA	ARC-C	ARC-E	HellaS.	NQ.	WinoG.	Avg.
Base Model: Qwen-2.5-7B										
Naive: MoEfication	55.44	93.80	72.31	33.07	40.78	72.01	49.07	8.20	58.64	53.70
+ Functional Grouping	62.16	93.80	76.33	37.76	44.28	73.19	53.24	13.60	62.19	57.39
+ Pairwise-ranked Routing	66.80	94.40	77.80	38.12	47.18	78.66	56.83	17.59	67.40	60.53
+ Sparsity Allocation	67.68	94.80	78.56	37.49	47.95	79.75	**57.38**	17.42	68.90	61.10
+ STE-based Adaptation	**69.47**	**96.00**	**79.16**	**42.51**	**50.60**	**81.10**	56.24	**20.03**	**71.67**	**62.98**
Base Model: LLaMA-3.1-8B										
Naive: MoEfication	50.44	94.00	77.09	32.43	39.68	70.37	51.02	10.58	67.72	54.81
+ Functional Grouping	51.75	94.30	76.66	34.67	40.44	72.39	52.66	9.92	68.98	55.75
+ Pairwise-ranked Routing	52.34	94.20	76.50	34.81	43.34	72.69	53.98	11.44	69.46	56.53
+ Sparsity Allocation	53.06	95.10	77.04	34.81	43.26	73.19	54.20	11.05	71.03	56.97
+ STE-based Adaptation	**56.99**	**95.80**	**77.75**	**35.31**	**45.90**	**78.45**	**55.88**	**13.93**	**70.30**	**58.92**

4.2 Main Results

We compare different Dense-to-MoE conversion methods on Qwen-2.5-7B and LLaMA-3.1-8B under multiple sparsity settings. Table 1 reports detailed benchmark results at two representative activation budgets (30% and 60% activated experts), together with the corresponding dense-model reference. Results under additional sparsity budgets (e.g., 40%, 50%, etc.), experiments on the larger Qwen-2.5-14B, and further training details are provided in Appendix B.

As shown in Table 1, ExpertNavigator consistently outperforms prior Dense-to-MoE baselines across all benchmarks under different sparsity budgets. On Qwen-2.5-7B, ExpertNavigator retains **88.9%** of the dense model's average benchmark score while activating only 30% of FFN parameters, and further preserves **97.8%** of the average score at 60% activation; it also exhibits a consistent advantage on LLaMA-3.1-8B, demonstrating high-fidelity Dense-to-MoE conversion.

Notably, with 60% FFN activation on Qwen-2.5-7B, ExpertNavigator's performance drops are mainly concentrated on knowledge-intensive tasks that benefit from more complete activation (e.g., MMLU), whereas most other benchmarks remain nearly lossless. In some cases (e.g., PIQA and ARC-C), ExpertNavigator even slightly surpasses the original dense model. These results not only reaffirm the substantial redundancy in dense FFNs [14, 64], but also indicate that ExpertNavigator largely preserves the functional organization of the original model. They further suggest that STE-based joint training enables continued post-conversion improvement with additional high-quality data, with the potential to even exceed the original dense baseline.

In Appendix B.3, we further compare the ExpertNavigator-converted model based on Qwen-2.5-7B with 60% FFN activation (approximately 4.5B activated parameters per token) against several contemporary open-source LLMs. The results show that ExpertNavigator (A4.5B) is highly competitive among models of similar activated capacity. The larger-model study in Appendix B.3 shows that ExpertNavigator also scales effectively to Qwen-2.5-14B, preserving **96.4%** of the dense model's average score under the same 60% activation budget and maintaining a clear advantage over prior Dense-to-MoE baselines. Together, these results indicate that the proposed conversion strategy is not only competitive at comparable activated capacity, but also robust to increased model scale.

In addition to quality, ExpertNavigator yields tangible inference-efficiency gains via sparse activation, consistent with prior Dense-to-MoE methods [47, 65]. Results in Appendix C.1 show that, at 60% FFN activation, ExpertNavigator reduces inference FLOPs by about 30% and achieves an end-to-end generation wall-clock speedup of over 15% compared to the original dense model.

4.3 Ablation Studies

ExpertNavigator constitutes a complete Dense-to-MoE conversion pipeline with multiple novel components. To assess the role of each component, we conduct extensive ablation studies that aim to (i) quantify the contribution of every module in ExpertNavigator and (ii) disentangle the effects of expert grouping and router-training objectives under controlled settings. All experiments use the same base models, activation budgets, training data, and evaluation suite.

Table 2 presents a progressive ablation starting from a naive MoEfication baseline (W_g-based co-activation expert grouping with soft-BCE router training) and incrementally adding our components.

Table 3: Ablation study of different expert grouping and router training methods. We compare each factor in a controlled setting by fixing the other (grouping or routing objective). Experiments are conducted on Qwen-2.5-7B with 60% expert activation.

ExpertGrouping	RouterTraining	MMLU	SciQ	PIQA	MathQA	ARC-C	ARC-E	HellaS.	NQ.	WinoG.	Avg.
W_g Cluster	Soft-BCE	61.33	92.40	75.19	33.63	40.96	71.21	51.89	12.08	59.12	55.31
Co-activation	Soft-BCE	55.44	**93.80**	72.31	33.07	40.78	72.01	49.07	8.20	58.64	53.70
FuncGrouping	Soft-BCE	**62.16**	**93.80**	**76.33**	**37.76**	**44.28**	**73.19**	**53.24**	**13.60**	**62.19**	**57.39**
FuncGrouping	MSE Regression	66.46	**94.60**	**77.80**	**39.06**	46.42	77.19	56.64	15.51	64.56	59.80
FuncGrouping	Soft-BCE	62.16	93.80	76.33	37.76	44.28	73.19	53.24	13.60	62.19	57.39
FuncGrouping	Pairwise Ranking	**66.80**	94.40	**77.80**	38.12	**47.18**	**78.66**	**56.83**	**17.59**	**67.40**	**60.53**

Each step yields consistent improvements, and the gains accumulate monotonically on both Qwen-2.5-7B and LLaMA-3.1-8B. In particular, replacing co-activation grouping with our functionally coherent grouping produces a stronger expert partition; replacing soft-BCE magnitude supervision with pairwise-ranked routing further improves expert selection. Adaptive layer-wise sparsity allocation brings additional gains under the same global budget. Notably, even without STE adaptation, this variant already outperforms all baselines in Table 1, achieving the best overall performance, and STE-based adaptation provides a further boost by improving model–router compatibility. The consistent upward trend across two models indicates that the benefits of ExpertNavigator are robust and not tied to a specific architecture.

Effect of expert grouping. To isolate the impact of expert construction, we compare three representative grouping strategies: (i) weight-based clustering on the gating matrix W_g, (ii) gate co-activation based grouping, and (iii) our functionally coherent grouping. To ensure a fair comparison, we fix the router training objective to the same soft BCE supervision used in prior work and keep all other settings identical. As shown in Table 3, our functional grouping consistently achieves the best performance. This result supports our motivation in Sec. 3.1: capturing full-neuron functional signals beyond the gate pathway is critical for constructing experts that preserve the original dense FFN behavior. We provide additional diagnostics based on cumulative errors of tail truncation (CETT) in Appendix B.4, further showing that our grouping better preserves dense FFN functionality.

Effect of router-training objective. We study router-learning objectives while holding expert grouping fixed to our functional partition. We compare (i) soft-BCE supervision for activation-magnitude prediction, (ii) MSE regression on expert output norms, and (iii) our pairwise rank distillation. Results in Table 3 show that pairwise ranking yields the strongest performance, consistently outperforming objectives that regress absolute scales. This finding aligns with our analysis in Sec. 3.2: Dense-to-MoE routing is ultimately a discrete hard top-k selection problem, and supervising relative expert ordering is better conditioned and more stable than fitting scale-variant absolute magnitudes.

Moreover, in Appendix B.5, we repeat the above ablations with coarser 64-neuron experts. The results show that our Expert Grouping and Router Training still maintain clear advantages and exhibit stronger robustness. Together, these ablations confirm that ExpertNavigator's gains arise from complementary improvements across multiple stages of the conversion pipeline. These components work in concert, ultimately enabling high-fidelity Dense-to-MoE conversion.

5 Conclusion, Limitations and Future Work

In this paper, we propose **ExpertNavigator**, a system for high-fidelity Dense-to-MoE conversion that targets the key bottlenecks of expert construction and router learning under hard top-k sparse activation. ExpertNavigator builds functionally coherent experts via contribution-based grouping, trains routers with a pairwise rank distillation objective that matches discrete top-k selection, and further improves the quality–efficiency trade-off through adaptive layer-wise sparsity allocation. Finally, we employ STE-based full-parameter adaptation to refine model–router compatibility under hard routing. Across Qwen-2.5-7B and LLaMA-3.1-8B, ExpertNavigator consistently outperforms prior Dense-to-MoE baselines over a range of activation budgets, achieving substantially smaller quality gaps to the dense models while enabling practical inference speedups.

Limitations and future work. Consistent with prior Dense-to-MoE methods, we adopt a fine-grained expert granularity (e.g., 16 neurons per expert). However, expert granularity involves a fundamental quality–efficiency trade-off: finer partitions weaken structured sparsity and may reduce practical speedup, whereas coarser partitions can distort the original dense activation patterns and widen the performance gap. Thus, identifying an optimal granularity remains an open question that warrants

further exploration. Moreover, although Dense-to-MoE conversion can yield substantial theoretical FLOPs reductions, these savings do not necessarily translate into proportional end-to-end latency improvements; achieving further real-world acceleration likely requires specialized sparse kernels and system-level optimizations tailored to fine-grained routing, which we leave for future work.

Acknowledgments and Disclosure of Funding

This work was supported in part by the Scientific Research Innovation Capability Support Project for Young Faculty (U40) of the Ministry of Education of China (SRICSPYF-ZY2025019).

References

[1] Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)*, pages 2357–2367, 2019.

[2] Saleh Ashkboos, Maximilian L Croci, Marcelo Gennari do Nascimento, Torsten Hoefler, and James Hensman. Slicegpt: Compress large language models by deleting rows and columns. *arXiv preprint arXiv:2401.15024*, 2024.

[3] Yoshua Bengio. Estimating or propagating gradients through stochastic neurons. *arXiv preprint arXiv:1305.2982*, 2013.

[4] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[5] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John Guttag. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

[6] Ruisi Cai, Yeonju Ro, Geon-Woo Kim, Peihao Wang, Babak Ehteshami Bejnordi, Aditya Akella, Zhangyang Wang, et al. Read-me: Refactorizing llms as router-decoupled mixture of experts with system co-design. *Advances in Neural Information Processing Systems*, 37: 116126–116148, 2024.

[7] Shuochen Chang, Tong Bai, Xiaofeng Zhang, Qianli Ma, Qingyang Liu, Zhaohe Liao, Yibo Miao, and Li Niu. Unlocking the black box of latent reasoning: An interpretability-guided approach to intervention. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 34019–34032, 2026.

[8] Shuochen Chang, Qingyang Liu, Shaobo Wang, Bingjie Gao, Qianli Ma, Haonan Zhao, Yibo Miao, Yulin Sun, Zelin Peng, Jiangtong Li, et al. Why struggle with continuous latents? interpretable discrete latent reasoning via rendered compression. *arXiv preprint arXiv:2606.29712*, 2026.

[9] Rewon Child. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[10] 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. *arXiv preprint arXiv:1803.05457*, 2018.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[12] Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in neural information processing systems*, 35:16344–16359, 2022.

[13] Gongfan Fang, Xinyin Ma, Mingli Song, Michael Bi Mi, and Xinchao Wang. Depgraph: Towards any structural pruning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16091–16101, 2023.

[14] 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.

[15] Determine Filters'Importance. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[16] Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International conference on machine learning*, pages 10323–10337. PMLR, 2023.

[17] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

[18] Shangqian Gao, Ting Hua, Reza Shirkavand, Chi-Heng Lin, Zhen Tang, Zhengao Li, Longge Yuan, Fangyi Li, Zeyu Zhang, Alireza Ganjdanesh, et al. Tomoe: Converting dense large language models to mixture-of-experts through dynamic structural pruning. *arXiv preprint arXiv:2501.15316*, 2025.

[19] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[20] Ethan He, Abhinav Khattar, Ryan Prenger, Vijay Korthikanti, Zijie Yan, Tong Liu, Shiqing Fan, Ashwath Aithal, Mohammad Shoeybi, and Bryan Catanzaro. Upcycling large language models into mixture of experts. *arXiv preprint arXiv:2410.07524*, 2024.

[21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[22] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. *Advances in neural information processing systems*, 29, 2016.

[23] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

[24] Mingyu Jin, Qinkai Yu, Jingyuan Huang, Qingcheng Zeng, Zhenting Wang, Wenyue Hua, Haiyan Zhao, Kai Mei, Yanda Meng, Kaize Ding, et al. Exploring concept depth: How large language models acquire knowledge and concept at different layers? In *Proceedings of the 31st international conference on computational linguistics*, pages 558–573, 2025.

[25] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[26] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998. URL https://doi.org/10.1137/S1064827595287997.

[27] Eldar Kurtic, Daniel Campos, Tuan Nguyen, Elias Frantar, Mark Kurtz, Benjamin Fineran, Michael Goin, and Dan Alistarh. The optimal bert surgeon: Scalable and accurate second-order pruning for large language models. *arXiv preprint arXiv:2203.07259*, 2022.

[28] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

[29] Ning Liao, Xiaoxing Wang, Zehao Lin, Weiyang Guo, Feng Hong, Shixiang Song, Geng Yu, Zihua Zhao, Sitao Xie, Longxuan Wei, et al. Innovator: Scientific continued pretraining with fine-grained moe upcycling. *arXiv preprint arXiv:2507.18671*, 2025.

[30] Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. Modegpt: Modular decomposition for large language model compression. *arXiv preprint arXiv:2408.09632*, 2024.

[31] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[32] Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Re, et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pages 22137–22176. PMLR, 2023.

[33] Xudong Lu, Qi Liu, Yuhui Xu, Aojun Zhou, Siyuan Huang, Bo Zhang, Junchi Yan, and Hongsheng Li. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. *arXiv preprint arXiv:2402.14800*, 2024.

[34] Yuqi Luo, Chenyang Song, Xu Han, Yingfa Chen, Chaojun Xiao, Xiaojun Meng, Liqun Deng, Jiansheng Wei, Zhiyuan Liu, and Maosong Sun. Sparsing law: Towards large language models with greater activation sparsity. *arXiv preprint arXiv:2411.02335*, 2024.

[35] Chi Ma, Mincong Huang, Ying Zhang, Chao Wang, Yujie Wang, Lei Yu, Chuan Liu, and Wei Lin. First activations matter: Training-free methods for dynamic activation in large language models. *arXiv preprint arXiv:2408.11393*, 2024.

[36] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

[37] Mikko I Malinen and Pasi Fränti. Balanced k-means for clustering. In *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pages 32–41. Springer, 2014.

[38] Xin Men, Mingyu Xu, Qingyu Zhang, Qianhao Yuan, Bingning Wang, Hongyu Lin, Yaojie Lu, Xianpei Han, and Weipeng Chen. Shortgpt: Layers in large language models are more redundant than you expect. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 20192–20204, 2025.

[39] Taishi Nakamura, Takuya Akiba, Kazuki Fujii, Yusuke Oda, Rio Yokota, and Jun Suzuki. Drop-upcycling: Training sparse mixture of experts with partial re-initialization. *arXiv preprint arXiv:2502.19261*, 2025.

[40] Zehua Pei, Lancheng Zou, Hui-Ling Zhen, Xianzhi Yu, Wulong Liu, Sinno Jialin Pan, Mingxuan Yuan, and Bei Yu. Cmoe: Converting mixture-of-experts from dense to accelerate llm inference. *arXiv preprint arXiv:2502.04416*, 2025.

[41] Xiaohan Qin, Victor Wang, Ning Liao, Cancheng Zhang, Xiangdong Zhang, Mingquan Feng, Jingzhi Wang, and Junchi Yan. sstoken: Self-modulated and semantic-aware token selection for llm fine-tuning. In *International Conference on Learning Representations*, volume 2026, pages 79347–79368, 2026.

[42] Xiaohan Qin, Xiangdong Zhang, Tuo Dai, Sunan Zou, Huaijin Wu, Yebin Yang, Zhuo Xia, Lin Yao, Yuliang Liu, and Junchi Yan. Multi-stream attention residuals. *arXiv preprint arXiv:2610.13630*, 2026.

[43] Qwen, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[44] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106, 2021.

[45] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

[46] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

[47] Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Simone Scardapane. Exploiting activation sparsity with dense to dynamic-k mixture-of-experts conversion. *Advances in Neural Information Processing Systems*, 37:43245–43273, 2024.

[48] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[49] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[50] Qwen Team et al. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2(3), 2024.

[51] Shaobo Wang, Xiangqi Jin, Ziming Wang, Jize Wang, Jiajun Zhang, Kaixin Li, Zichen Wen, Zhong Li, Conghui He, Xuming Hu, et al. Data whisperer: Efficient data selection for task-specific llm fine-tuning via few-shot in-context learning. *arXiv preprint arXiv:2505.12212*, 2025.

[52] Yudong Wang, Zixuan Fu, Jie Cai, Peijun Tang, Hongya Lyu, Yewei Fang, Zhi Zheng, Jie Zhou, Guoyang Zeng, Chaojun Xiao, et al. Ultra-fineweb: Efficient data filtering and verification for high-quality llm training data. *arXiv preprint arXiv:2505.05427*, 2025.

[53] Xiuying Wei, Skander Moalla, Razvan Pascanu, and Caglar Gulcehre. Building on efficient foundations: Effective training of llms with structured feedforward layers. *Advances in Neural Information Processing Systems*, 37:4689–4717, 2024.

[54] Johannes Welbl, Nelson F Liu, and Matt Gardner. Crowdsourcing multiple choice science questions. *arXiv preprint arXiv:1707.06209*, 2017.

[55] Huaijin Wu, Lianqiang Li, Hantao Huang, Yi Tu, Jihang Zhang, Minghui Yu, and Junchi Yan. HShare: Fast LLM decoding by hierarchical key-value sharing. In *International Conference on Learning Representations*, volume 2025, pages 89412–89428, 2025.

[56] Huaijin Wu, Bingrui Li, Yebin Yang, Yi Tu, Zhanpeng Zhou, Jianfei Chen, and Junchi Yan. Achieving low-bit Muon through subspace preservation and grid quantization. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=g2l9bg9DWx.

[57] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*, 2022.

[58] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[59] Yebin Yang, Huaijin Wu, Fu Guo, Lin Yao, Xiaohan Qin, Jingzhi Wang, Debing Zhang, and Junchi Yan. Jtok: On token embedding as another axis of scaling law via joint token self-modulation. *arXiv preprint arXiv:2602.00800*, 2026.

[60] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*, 2019.

[61] Xiangdong Zhang, Xiaohan Qin, Sunan Zou, Tuo Dai, Xiaoming Shi, Huaijin Wu, Yebin Yang, Zhuo Xia, Shaofeng Zhang, Lin Yao, et al. xhc: Expanded hyper-connections. *arXiv preprint arXiv:2607.14530*, 2026.

[62] Xiangdong Zhang, Debing Zhang, Shaofeng Zhang, Xiaohan Qin, Yu Cheng, and Junchi Yan. Nitp: Next implicit token prediction for llm pre-training. *arXiv preprint arXiv:2605.24956*, 2026.

[63] Zhengyan Zhang, Yankai Lin, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Moefication: Transformer feed-forward layers are mixtures of experts. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 877–890, 2022.

[64] Zhengyan Zhang, Yixin Song, Guanghui Yu, Xu Han, Yankai Lin, Chaojun Xiao, Chenyang Song, Zhiyuan Liu, Zeyu Mi, and Maosong Sun. Relu^2 wins: Discovering efficient activation functions for sparse llms. *arXiv preprint arXiv:2402.03804*, 2024.

[65] Haizhong Zheng, Xiaoyan Bai, Xueshen Liu, Zhuoqing Morley Mao, Beidi Chen, Fan Lai, and Atul Prakash. Learn to be efficient: Build structured sparsity in large language models. *Advances in Neural Information Processing Systems*, 37:101969–101991, 2024.

A Definitions of Notations

Due to the numerous concepts and symbols used throughout our Dense-to-MoE conversion pipeline, we provide definitions of the notations in Table 4 to assist readers in better understanding the content.

Table 4: Definitions of notations in this paper.

Variable	Definition
d	model hidden dimension
d_h	FFN intermediate (expansion) dimension
G	number of neurons per expert (expert granularity, e.g., 16)
K	number of experts per layer, $K = d_h/G$
k	top-k activated experts per token (hard routing)
$x \in \mathbb{R}^d$	input token representation to the FFN
$\mathcal{D}$	dataset / token distribution for estimating grouping and allocation statistics
$\mathrm{FFN}(\cdot)$	dense feed-forward network mapping
$W_g \in \mathbb{R}^{d \times d_h}$	gating projection matrix in SwiGLU FFN
$W_{\mathrm{up}} \in \mathbb{R}^{d \times d_h}$	up projection matrix in SwiGLU FFN
$W_{\mathrm{down}} \in \mathbb{R}^{d_h \times d}$	down projection matrix in SwiGLU FFN
h	SwiGLU intermediate activation, $h = \mathrm{SiLU}(xW_g) \odot (xW_{\mathrm{up}})$
$\odot$	element-wise (Hadamard) product
$\pi(\cdot)$	expert partitioning (grouping) operator
$E_i(\cdot)$	function of expert i induced by FFN partitioning
$f_j(x)$	functional computation (contribution) of neuron j on input x
$c_j(x)$	neuron contribution strength, $c_j(x) = \|f_j(x)\|_2$
S_{ij}	functional coherence / co-contribution between neurons i and j
$G = (V, E)$	weighted neuron graph for grouping, with edge weight S_{ij}
$\mathcal{C}_e$	neuron set (cluster) corresponding to expert e
$\mathrm{Router}(\cdot)$	lightweight routing network
$\mathbf{z} \in \mathbb{R}^K$	router logits, $\mathbf{z} = \mathrm{Router}(x)$
$\mathbf{s} \in \mathbb{R}^K$	teacher scores over experts for a token
s_i	teacher score of expert i, typically $s_i = \|E_i(x)\|_2$
$\mathcal{P}$	set of ordered expert pairs (i, j) with $s_i > s_j$
$\mathcal{L}_{\mathrm{rank}}$	pairwise ranking loss for router training
ℓ	layer index
CV_ℓ	coefficient of variation of expert-score distribution in layer ℓ
w_ℓ	layer weight derived from CV_ℓ
p_ℓ	normalized layer allocation weight after softmax
$\mathrm{TopKMask}(\mathbf{z}, k)$	hard top-k expert selection operator
$\mathbf{m} \in \{0, 1\}^K$	hard routing mask indicating selected experts
τ	temperature for softmax in STE
$\mathbf{p}$	soft expert probabilities, $\mathbf{p} = \mathrm{softmax}(\mathbf{z}/\tau)$
$\hat{\mathbf{m}}$	STE gate, $\hat{\mathbf{m}} = \mathbf{m} + (\mathbf{p} - \mathrm{StopGrad}(\mathbf{p}))$

B Additional Experimental Results

B.1 Training Details

Within the broader landscape of LLM pretraining [42, 56, 59, 61, 62] and post-training and inference [7, 8, 41, 55], our experiments focus on adapting pretrained dense models. In our pipeline, training is required in Stage 2 (router training) and Stage 4 (STE-based model–router adaptation). We report detailed hyperparameters below. Following prior works [47, 63], the router at each layer is implemented as a lightweight two-layer MLP with a SiLU nonlinearity:

$$\text{Router}(x) = W_2\,\sigma(W_1 x)\,, \tag{12}$$

where $W_1 \in \mathbb{R}^{d_r \times d}$ and $W_2 \in \mathbb{R}^{K \times d_r}$. We use a small hidden dimension $d_r \ll d$, and set $d_r = 256$ in all experiments. Here K is the number of experts per layer. We follow the fine-grained setting where each expert contains 16 neurons; under this setting, Qwen-2.5-7B has $K = 1184$ experts per layer and LLaMA-3.1-8B has $K = 896$ experts per layer. Notably, our clustering procedure is efficient: the co-contribution computation S_{ij} is fully vectorized and benefits from GPU parallelism, requiring < 3 minutes on 1,000 sequences of length 2,048, and the overall clustering (including greedy equal-size assignment) completes in < 5 minutes. By comparison, balanced k-means on W_g takes more than 1 hour due to the large hidden dimension d_h, while constructing and clustering the gated co-activation graph can exceed 8 hours because it relies on sparse top-k selection and indexing operations.

We use the open-source Ultra-FineWeb pretraining corpus [52] for both router training and STE-based adaptation. The sequence length is 2,048 and the global batch size is 16. We use a learning rate of 5×10^{-4} for Stage 2 (router-only training) and 5×10^{-6} for Stage 4 (joint model–router adaptation). For Stage 2, we sample 2,048 expert pairs per token to construct the pairwise ranking loss. For Stage 4, we set the STE temperature to $\tau = 3.0$. These hyperparameters are also validated via ablations; we find that modest variations in these values have a relatively minor impact compared to the effects of our core components. To ensure fair comparisons with prior Dense-to-MoE methods, the entire conversion pipeline uses fewer than 1B training tokens in total. The project page is available at `https://github.com/jianke0604/ExpertNavigator`.

Pair Sampling Strategy and Analysis. For pairwise-ranked router training, we adopt a RankNet-style sampling strategy. Given the teacher scores $\mathbf{s}$ over experts for each token, we first sort experts by their teacher scores and use the top-512 experts as the positive pool, while the remaining experts form the negative pool. For each positive expert, we sample four negative experts, resulting in 2,048 ordered pairs per token. This construction focuses the router supervision on separating high-contribution experts from low-contribution experts, which is directly aligned with the hard top-k selection used during inference.

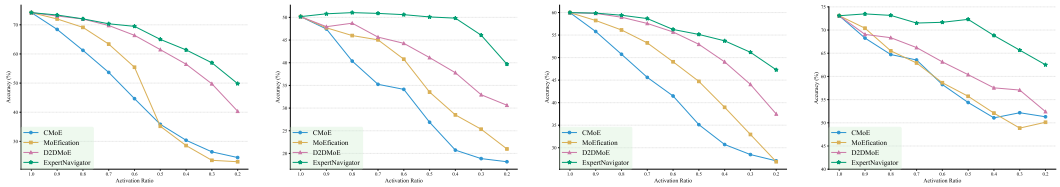
We further examine the sensitivity of this sampling strategy by reducing the number of sampled pairs to 1,024. Specifically, we compare two variants: (i) sampling positives from the top-256 experts, which provides high-confidence positive examples, and (ii) sampling positives from the bottom-256 experts within the original top-512 pool, which emphasizes boundary positives closer to the decision margin. The negative pool is kept unchanged for both variants. Table 5 reports the results on Qwen-2.5-7B under 60% expert activation.

Table 5: Sensitivity analysis of pair sampling strategies on Qwen-2.5-7B under 60% expert activation.

Pairs	**2,048**	**1,024 (top)**	**1,024 (boundary)**
Avg.	**62.98**	62.81	61.37

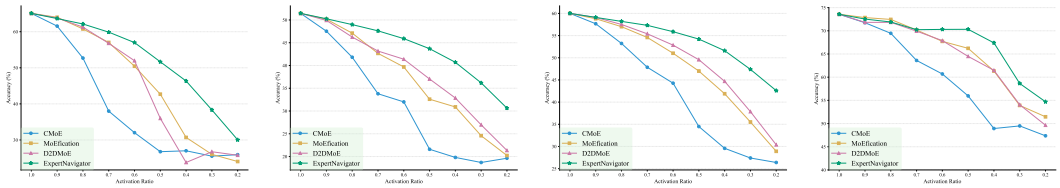
The results show that reducing the pair count from 2,048 to 1,024 causes only a negligible drop when positives are sampled from high-confidence top-ranked experts. In contrast, using boundary positives leads to a larger degradation. This suggests that the router training is not highly sensitive to the exact number of sampled pairs, but benefits from clear ranking signals that provide reliable separation between important and unimportant experts.

Converting Layer-wise Ratios to Integer Top-k. Given normalized allocation weights $\{p_\ell\}_{\ell=1}^{L}$ with $\sum_{\ell=1}^{L} p_\ell = 1$ and a global activation ratio α (e.g., $\alpha = 0.6$), the per-layer activation ratio is computed as $p_\ell \cdot \alpha L$, so the layer-wise ratios sum to the global budget αL. The number of activated



(a) MMLU (b) ARC-C (c) HellaSwag (d) WinoGrande

Figure 4: Extended results on Qwen-2.5-7B under varying expert activation ratios across four benchmarks.



(a) MMLU (b) ARC-C (c) HellaSwag (d) WinoGrande

Figure 5: Extended results on LLaMA-3.1-8B under varying expert activation ratios across four benchmarks.

experts at layer ℓ is then obtained by

$$k_\ell = \lfloor K \cdot p_\ell \cdot \alpha L \rfloor , \tag{13}$$

where K denotes the total number of experts in each layer. This converts the continuous layer-wise allocation into an integer top-k routing budget for each layer.

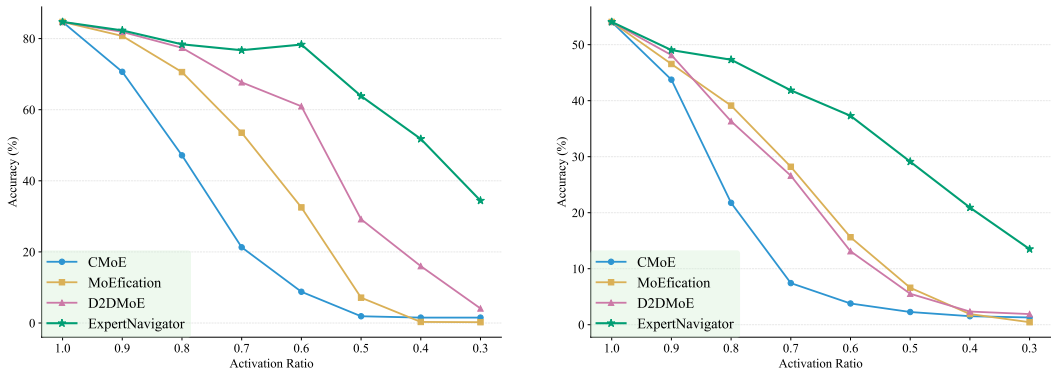
B.2 Extended Results under Varying Activation Ratios

To complement the main results reported at two representative budgets in Table 1, we further evaluate performance under a wide range of activation ratios. Figures 4 and 5 plot accuracy curves on four representative benchmarks (MMLU, ARC-C, HellaSwag, and WinoGrande) as the per-layer expert activation ratio varies over $\{0.2, 0.3, \ldots, 1.0\}$. We compare CMoE, MoEfication, D2DMoE, and ExpertNavigator under the same activation ratios. Note that LTE does not directly support manually specifying a target activation ratio; instead, its sparsity emerges implicitly from training hyperparameters. The LTE results in Table 1 correspond to models whose hyperparameters are tuned to yield approximate 30% and 60% activation, and we omit LTE from the continuous curves for a controlled comparison.

Across both model families and all four benchmarks, ExpertNavigator exhibits a consistent advantage throughout the entire sparsity range. In particular, ExpertNavigator degrades more gracefully as activation decreases, maintaining markedly stronger performance in low-budget regimes (e.g., 0.2–0.4) where several baselines drop sharply. This trend is especially evident on more sparsity-sensitive tasks (e.g., MMLU), suggesting that our functionally coherent grouping and pairwise-ranked routing yield more reliable top-k selection under aggressive sparsity. Meanwhile, for benchmarks that are less dependent on full activation (e.g., ARC-C, WinoGrande), the gap remains small even at moderate budgets, and ExpertNavigator often stays closest to the dense reference. The consistent dominance of our curves on both Qwen-2.5-7B and LLaMA-3.1-8B further indicates that ExpertNavigator's improvements are robust and transfer across model families.

Performance on Complex Reasoning Tasks. Beyond the general benchmarks in Table 1, we further evaluate the converted models on GSM8K-CoT [11], a substantially more challenging setting that emphasizes multi-step arithmetic reasoning and coherent intermediate derivations. We adopt 8-shot prompting to elicit chain-of-thought solutions, and report final answer accuracy. Fig. 6 summarizes performance as a function of the FFN activation ratio, providing a stress test of how well different conversion methods preserve reasoning-relevant computation under increasingly aggressive sparsity.

As shown in the figure, several baselines degrade sharply as the activation ratio decreases. In the low-budget regime (e.g., 20%–30% activation), some methods even collapse to near-zero accuracy, suggesting that their routed sparse computation fails to recover the dense model's core reasoning behaviors once only a small fraction of experts is active. This failure mode is particularly concerning for Dense-to-MoE conversion, since complex reasoning requires reliably composing multiple intermediate steps rather than only matching shallow pattern completion. In contrast, ExpertNavigator exhibits markedly more graceful degradation and maintains a consistent advantage across the entire sparsity range, retaining competitive accuracy even at low activation ratios. These results indicate that ExpertNavigator better preserves the functional structure needed for multi-step reasoning under hard top-k routing.



(a) Qwen-2.5-7B (b) LLaMA-3.1-8B

Figure 6: Performance on GSM8K-CoT under varying FFN activation ratios.

B.3 Scalability and Open-Source Model Comparison

To examine whether the advantages of ExpertNavigator persist at a larger model scale, we further conduct Dense-to-MoE conversion on Qwen-2.5-14B under a 60% expert activation budget. As shown in Table 6, ExpertNavigator maintains a clear advantage over prior conversion baselines and preserves **96.4%** of the dense model's average benchmark score. The performance gap over D2DMoE is larger than that observed on Qwen-2.5-7B, suggesting that conversion quality becomes increasingly important as depth grows and routing errors can accumulate across more layers.

Table 6: Performance comparison of different Dense-to-MoE methods on Qwen-2.5-14B under a 60% activation budget. The best results are highlighted in **bold**. We also report the ratio of each method's average benchmark score to that of the original dense model.

Methods	Act.	MMLU	SciQ	PIQA	MathQA	ARC-C	ARC-E	HellaS.	NQ.	WinoG.	Avg.
Base Model: Qwen-2.5-14B											
Dense	100%	79.78	96.80	81.28	52.73	56.23	82.62	63.36	30.61	75.14	68.73 100%
CMoE	60%	52.10	93.60	69.18	34.76	38.21	62.08	44.78	5.31	61.70	51.30 74.6%
MoEfication	60%	64.46	94.80	76.17	41.71	45.73	75.72	54.09	11.26	63.30	58.58 85.2%
LTE	60%	54.62	93.20	74.97	35.82	44.97	75.10	54.85	9.77	65.10	56.49 82.2%
D2DMoE	60%	66.34	95.00	76.48	43.92	48.29	77.48	57.08	20.06	65.82	61.16 89.0%
ExpertNavigator	60%	**73.54**	**96.20**	**79.65**	**48.27**	**55.97**	**83.12**	**59.81**	**25.33**	**74.51**	**66.27** 96.4%

Comparison with Other Open-Source Models. To further contextualize the strength of the converted model, we compare ExpertNavigator against a set of competitive open-source LLMs. Specifically, we evaluate the Dense-to-MoE model converted from Qwen-2.5-7B with a 60% expert activation ratio; under this setting, each token activates approximately 4.5B parameters. The open-source models we compare against include Qwen-2.5-3B [50] from the same family, the more recent and stronger Qwen3-4B [58] and Gemma-3-4B [48], as well as LLaMA-3.2-3B [19] from the LLaMA family, the larger LLaMA-3.1-8B [19], and Mistral-7B-v0.3 [23].

As shown in Table 7, ExpertNavigator remains highly competitive among these strong open-source models. It achieves the second-best average score overall, surpassed only by the substantially larger LLaMA-3.1-8B, and outperforms similarly sized models such as Qwen3-4B and Gemma-3-4B. These results further validate that ExpertNavigator can preserve a large fraction of the original dense model's capability under sparse activation, yielding a model that is not only efficient in activated computation but also competitive in absolute quality relative to widely used open-source alternatives.

Table 7: Benchmark results across different models. The best results are highlighted with **best**, the second-best with second, and the third-best with third.

Models	MMLU	SciQ	PIQA	MathQA	ARC-C	ARC-E	HellaS.	NQ.	WinoG.	Avg.
Llama-3.2-3B	56.45	95.70	76.66	35.08	42.15	74.54	55.30	21.52	69.85	58.58
Qwen-2.5-3B	65.55	95.90	78.18	36.92	44.97	77.44	55.02	19.00	68.11	60.12
Gemma3-4B	59.61	**96.30**	78.89	39.93	51.37	81.36	56.75	21.80	69.38	61.71
Llama-3.1-8B	65.08	96.20	80.03	39.53	**51.45**	**81.57**	60.01	27.70	**73.56**	**63.90**
Mistral-7B-v0.3	62.33	95.90	**80.30**	35.54	48.98	79.50	**60.89**	**28.75**	73.40	62.84
Qwen3-4B	**70.03**	95.50	75.08	**46.60**	50.34	80.47	52.21	15.24	65.27	61.19
ExpertNavigator(A4.5B)	69.47	96.00	79.16	42.51	50.60	81.10	56.24	20.03	71.67	62.98

B.4 Functional Coherence Analysis with CETT

We further diagnose expert grouping quality before introducing any router. Following prior sparse-activation analyses [34, 35, 64], we use cumulative errors of tail truncation (CETT) to measure how much FFN output is distorted when low-contribution experts are removed. For an input x, let D denote the set of tail experts under a given truncation threshold. We compute

$$\mathrm{CETT}(x) = \frac{\left\|\sum_{i\in D} E_i(x)\right\|_2}{\|\mathrm{FFN}(x)\|_2}, \qquad D = \{i \mid \|E_i(x)\|_2 < \epsilon\}. \tag{14}$$

Lower CETT indicates that the discarded tail experts contribute less to the original dense FFN output, implying a cleaner separation between high- and low-contribution expert groups.

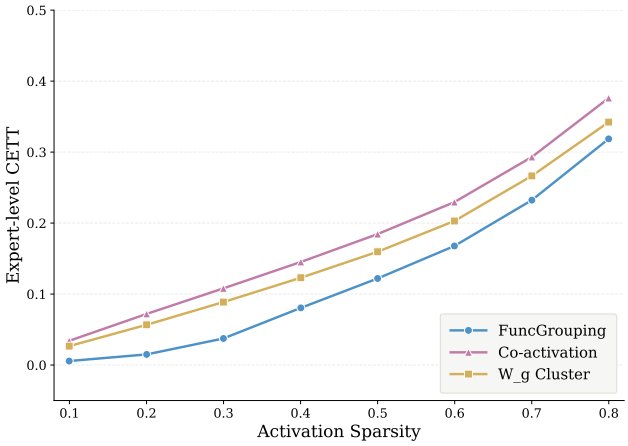


Figure 7: CETT comparison of different grouping strategies on Qwen-2.5-7B. Lower CETT indicates smaller FFN reconstruction error after tail truncation.

As shown in Fig. 7, our functionally coherent grouping consistently yields lower CETT across sparsity levels than W_g clustering and gate co-activation grouping. This result provides direct evidence that our grouping better preserves the functional organization of the original dense FFN: low-contribution neurons tend to be grouped together, so removing tail experts introduces smaller output distortion. This diagnostic is consistent with the downstream ablation results in Table 3.

B.5 Robustness Across Expert Granularities

The main experiments use a fine-grained setting where each expert contains 16 neurons. To test robustness to expert granularity, we additionally evaluate a coarser 64-neuron granularity on Qwen-

2.5-7B at 60% expert activation. This setting creates larger experts and therefore makes each routing decision more consequential, providing a stress test for both expert construction and router training.

Table 8: Ablation study of different expert grouping and router training methods under 64-neuron expert granularity. Experiments are conducted on Qwen-2.5-7B with 60% expert activation. ΔAvg. = $\text{Avg}_{64} - \text{Avg}_{16}$; **bold green** denotes the smallest drop in each group.

ExpertGrouping	RouterTraining	MMLU	SciQ	PIQA	MathQA	ARC-C	ARC-E	HellaS.	NQ.	WinoG.	Avg.	ΔAvg.
W_g Cluster	Soft-BCE	58.61	92.00	72.96	**35.41**	40.53	71.21	50.95	9.87	57.70	54.36	−0.95
Co-activation	Soft-BCE	53.48	**93.20**	73.07	30.26	40.02	70.41	47.57	7.91	57.62	52.62	−1.08
FuncGrouping	Soft-BCE	**62.06**	92.80	**76.39**	35.28	**42.41**	**74.62**	**52.61**	**11.31**	**63.85**	**56.81**	**−0.58**
FuncGrouping	MSE Regression	64.58	93.80	78.18	38.46	43.77	76.28	55.52	14.10	64.96	58.85	−0.95
FuncGrouping	Soft-BCE	62.06	92.80	76.39	35.28	42.41	74.62	52.61	11.31	63.85	56.81	−0.58
FuncGrouping	Pairwise Ranking	**65.98**	**94.20**	**78.29**	**38.96**	**46.10**	**77.95**	**55.98**	**17.13**	**66.27**	**60.10**	**−0.43**

As shown in Table 8, the conclusions from the 16-neuron setting remain consistent under coarser experts. Functional grouping still outperforms W_g clustering and co-activation grouping when the router objective is fixed, and pairwise ranking remains the strongest router-training objective when the grouping is fixed. The smaller ΔAvg. values indicate that our components are also more robust to the performance degradation caused by coarser expert granularity.

C Inference Efficiency Analysis

C.1 Comparison of FLOPs between standard FFN layer and ExpertNavigator

To quantify the computational efficiency of the ExpertNavigator-converted MoE relative to a standard dense FFN, we estimate the floating-point operations (FLOPs) of the two modules. Following common practice [12, 25], we ignore the negligible cost of biases, nonlinearities (e.g., SiLU), and element-wise operations, and focus on the matrix multiplications that dominate runtime.

Dense SwiGLU FFN. A standard SwiGLU FFN computes

$$h = \text{SiLU}(xW_g) \odot (xW_{\text{up}}), \qquad \text{FFN}(x) = hW_{\text{down}}, \tag{15}$$

where $x \in \mathbb{R}^d$, $W_g, W_{\text{up}} \in \mathbb{R}^{d\times d_h}$, and $W_{\text{down}} \in \mathbb{R}^{d_h\times d}$. We count FLOPs as multiply-plus-add operations; thus a matrix multiplication of shape $(m \times k) \cdot (k \times n)$ costs $2mkn$ FLOPs. The FLOPs are dominated by three matrix multiplications: two projections from d to d_h, and one projection from d_h back to d. Thus, the per-token FLOPs of the dense FFN can be estimated as

$$C_{\text{dense}} \;=\; 2d\,d_h \;+\; 2d\,d_h \;+\; 2d_h\,d \;=\; 6\,d\,d_h. \tag{16}$$

Dynamic Expert Activation. In the ExpertNavigator-converted MoE, the intermediate dimension d_h is partitioned into $K = d_h/G$ experts, each containing G neurons. For a given token, the router produces logits $\mathbf{z} = \text{Router}(x) \in \mathbb{R}^K$ and activates k experts via $\text{TopKMask}(\mathbf{z}, k)$, so that only kG intermediate neurons participate in the FFN computation. Consequently, the FLOPs of the sparse SwiGLU FFN path scale proportionally to the activated width kG. Counting FLOPs as multiply-plus-add operations, the three dominant matrix multiplications cost

$$C_{\text{sparse-ffn}} \;=\; 2d\,(kG) \;+\; 2d\,(kG) \;+\; 2(kG)\,d \;=\; 6\,d\,G\,k. \tag{17}$$

In addition, the routing network introduces overhead. For a lightweight two-layer router with input dimension d, hidden dimension d_r, and output dimension K (cf. Eq. (12)), its FLOPs can be estimated as

$$C_{\text{router}} \;=\; 2d\,d_r \;+\; 2d_r\,K. \tag{18}$$

Therefore, the total per-token FLOPs of ExpertNavigator-converted MoE is

$$C_{\text{ExpertNavigator}} \;=\; C_{\text{sparse-ffn}} + C_{\text{router}} \;=\; 6\,d\,G\,k \;+\; 2d\,d_r \;+\; 2d_r\,K, \qquad K = \frac{d_h}{G}. \tag{19}$$

Relative cost. Finally, the FLOPs ratio between ExpertNavigator and the dense FFN is

$$\frac{C_{\text{ExpertNavigator}}}{C_{\text{dense}}} = \frac{6\,d\,G\,k + 2d\,d_r + 2d_r\,K}{6\,d\,d_h} \tag{20}$$

$$= \frac{k}{K} \;+\; \frac{d_r(d+K)}{3\,d\,d_h}, \qquad K = \frac{d_h}{G}. \tag{21}$$

Eq. (21) shows that ExpertNavigator reduces FFN computation approximately in proportion to the activation ratio k/K, while the router adds only a small additional overhead since $d_r \ll d$.

Concrete instantiation. For Qwen2.5-7B, we have $d = 3584$, $d_h = 18944$, $G = 16$ (thus $K = d_h/G = 1184$), and set the router hidden dimension to $d_r = 256$. Under an activation ratio of $k/K = 0.6$, Eq. (21) gives

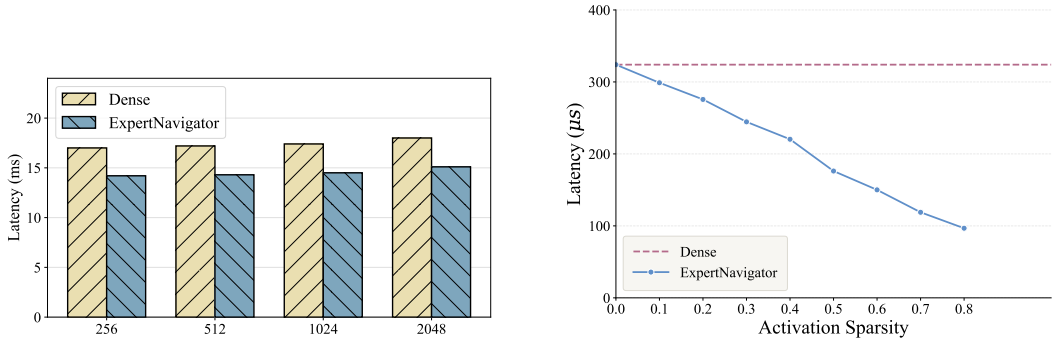
$$\frac{C_{\text{ExpertNavigator}}}{C_{\text{dense}}} \approx 0.6 + \frac{256(3584 + 1184)}{3 \cdot 3584 \cdot 18944} \approx 0.606, \tag{22}$$

corresponding to a $\approx 1.65\times$ FLOPs speedup for the FFN module (i.e., $\approx 39.4\%$ fewer FFN FLOPs). We further estimate the end-to-end per-token FLOPs for autoregressive decoding. Concretely, under $k/K = 0.6$ and a sequence length of 2048, ExpertNavigator achieves an end-to-end FLOPs speedup of approximately $1.43\times$ (about 30.0% fewer total FLOPs) compared to the dense model.

C.2 End-to-End Generation Latency Analysis

Consistent with prior Dense-to-MoE works [47, 63], ExpertNavigator induces highly structured sparsity via expert grouping. After conversion, we first permute FFN neurons so that neurons belonging to the same expert are stored contiguously, reducing memory-access overhead during sparse activation. Moreover, once the intermediate neurons to be activated are determined, the CUDA kernel only needs to load the corresponding rows/columns of W_g, W_{up}, and W_{down} from GPU global memory into on-chip memory to compute the dot products, thereby reducing both memory I/O and arithmetic cost. Concretely, following [65], we implement a custom Triton kernel tailored to this structured sparsity to translate theoretical savings into wall-clock speedups.

We report single-token end-to-end generation latency on Qwen-2.5-7B for the original dense model and the ExpertNavigator-converted MoE model with 60% expert activation under varying sequence lengths, measured on a single NVIDIA H200 GPU. As shown in Fig. 8, at 60% activation, ExpertNavigator reduces end-to-end generation latency by over 15% compared to the dense baseline. We also visualize FFN module latency under varying activation ratios in Fig. 8b, which shows the expected monotonic latency reduction as fewer experts are activated. We expect this speedup can be further improved with more optimized sparse Triton kernels and system-level implementations.



(a) End-to-end generation latency

(b) FFN module latency under varying sparsity

Figure 8: Inference latency analysis. We report end-to-end generation latency under different sequence lengths and FFN module latency under varying activation ratios.

D Baselines

D.1 MoEfication [63]

Expert Grouping. MoEfication follows the tensor-parallel style FFN slicing: given the number of experts K and a partition operator $\pi(\cdot)$, it partitions the d_h hidden neurons into K equal-size groups $\{\mathcal{C}_e\}_{e=1}^{K}$ with $|\mathcal{C}_e| = d_h/K$, yielding per-expert parameter blocks $\{W_g^{(e)}, W_{\text{up}}^{(e)}, W_{\text{down}}^{(e)}\}$. Instead of clustering W_g in the parameter space, MoEfication uses a **co-activation partition** strategy: it computes gate activations on training samples $x \sim \mathcal{D}$ (e.g., from $\text{SiLU}(xW_g)$) and builds a neuron

co-activation graph, where each node is a neuron and each edge weight measures how often two neurons are co-activated under the data distribution. Then, under the equal-size constraint, it applies graph partitioning (i.e., METIS [26]) to maximize within-group co-activation (equivalently, minimize the cut), producing expert assignment indices for each layer.

Router Training. MoEfication derives soft router targets from the gate branch. Given a token embedding x, it first clamps negative gate activations and then aggregates the positive responses within each expert:

$$t_e(x) = \sum_{j \in \mathcal{C}_e} \max\big(\mathrm{SiLU}\big((xW_g)_j\big), 0\big)\,. \tag{23}$$

The per-expert targets are then normalized independently for each token by the maximum expert score,

$$\tilde{t}_e(x) = \frac{t_e(x)}{\max(\max_{e'} t_{e'}(x), \epsilon)}, \tag{24}$$

where ϵ is a small constant used to clamp the denominator for numerical stability. This max-per-token normalization maps the teacher targets into $[0, 1]$ while preserving their relative activation strengths.

The router outputs per-expert logits $\{z_e(x)\}_{e=1}^K$ and is trained to predict the normalized soft targets $\{\tilde{t}_e(x)\}_{e=1}^K$ with a binary cross-entropy-with-logits objective,

$$\mathcal{L}_{\mathrm{BCE}}(x) = \frac{1}{K} \sum_{e=1}^{K} \Big[-\tilde{t}_e(x) \log \sigma(z_e(x)) - \big(1 - \tilde{t}_e(x)\big) \log\big(1 - \sigma(z_e(x))\big) \Big], \tag{25}$$

which aligns routing with the normalized gate-induced expert activity and enables input-dependent sparse expert activation at inference time.

D.2 D2DMoE [47]

Expert Grouping. D2DMoE constructs experts by clustering the gate matrix W_g in the parameter space [37]. Concretely, given the number of experts K, it applies a clustering algorithm to the d_h gate neurons (i.e., columns of W_g) and obtains an equal-size partition $\{\mathcal{C}_e\}_{e=1}^K$ with $|\mathcal{C}_e| = d_h/K$. The partition indices then induce the corresponding expert parameter blocks $\{W_g^{(e)}, W_{\mathrm{up}}^{(e)}, W_{\mathrm{down}}^{(e)}\}$, and define the e-th expert as

$$E_e(x) \;=\; \Big(\mathrm{SiLU}(xW_g^{(e)}) \odot (xW_{\mathrm{up}}^{(e)})\Big)\, W_{\mathrm{down}}^{(e)}. \tag{26}$$

Router Training. D2DMoE trains the router by using each expert's *output magnitude* as the routing target. Given an input token embedding x, it first computes each expert output $E_e(x) \in \mathbb{R}^d$ under the fixed expert partition, and defines a soft target

$$t_e(x) \;:=\; \|E_e(x)\|_2, \tag{27}$$

i.e., the L_2 norm of the expert output. The router outputs per-expert scores $\{p_e(x)\}_{e=1}^K$ and is trained to regress $\{t_e(x)\}_{e=1}^K$ by minimizing

$$\mathcal{L}_{\mathrm{MSE}}(x) = \frac{1}{K} \sum_{e=1}^{K} \big(p_e(x) - t_e(x)\big)^2, \tag{28}$$

so experts with larger output norms receive higher routing weights, enabling input-dependent sparse expert activation at inference time.

D.3 CMoE [40]

Expert grouping. Given an FFN with input $\mathbf{x} \in \mathbb{R}^d$ and intermediate size d_h, CMoE partitions neurons into K experts of size $m = d_h/K$, including K_s shared experts and K_r routed experts ($K_s + K_r = K$). Using a small calibration set, CMoE computes neuron-wise activation markers $\mathbf{a} = [a_1, \ldots, a_{d_h}]$ by selecting the K_a neurons with the largest $|h_i|$ per token and setting their

indicators to 1 (others 0), where $\mathbf{h} \in \mathbb{R}^{d_h}$ is the FFN hidden state. Neurons with the highest activation rates are collected into S_{K_s} to form shared experts E^s by slicing $(W_{\text{up}}, W_g, W_{\text{down}})$ accordingly. The remaining neurons are clustered into K_r routed experts with balanced constraints: CMoE picks K_r high-activation centroids $\{\hat{\mathbf{c}}_j\}_{j=1}^{K_r}$, computes distances $d_{i,j} = \|\mathbf{c}_i - \hat{\mathbf{c}}_j\|_2$ between neuron activation-pattern vectors $\mathbf{c}_i$ and centroids, and solves a balanced assignment to obtain index sets $\{S_{K_r,p}\}_{p=1}^{K_r}$, which define routed experts $\{E_p^r\}_{p=1}^{K_r}$ by the same weight slicing rule.

Sparse forward inference. The resulting MoE-FFN is

$$F_{\text{MoE}}(\mathbf{x}) = E^s(\mathbf{x}) + \sum_{i=1}^{K_r} g_i \, E_i^r(\mathbf{x}), \tag{29}$$

where $g_i \in \{0, 1\}$ activates the top-k routed experts. CMoE constructs the router G *training-free* using representative neurons: for each routed expert j, we select R_j as the neuron closest to centroid $\hat{\mathbf{c}}_j$ and set $S_R = \{R_1, \ldots, R_{K_r}\}$. The router scores are computed as $\mathbf{s} = G(\mathbf{x}) = [s_1, \ldots, s_{K_r}]$ using only the representative weights (W_{up}^R, W_g^R), and we set $g_i = 1$ iff $s_i \in \text{TopK}(\{s_j\}_{j=1}^{K_r}, k)$ (otherwise 0), yielding input-dependent sparse activation while preserving the dense parameters.

CMoE optionally makes routing differentiable by applying a zero-initialized learnable scale $\mathbf{u}$ to softened scores $\mathbf{s}' = \text{Softmax}(\mathbf{s})$ and selecting experts by $\text{TopK}(\mathbf{s}' + \mathbf{b}, k)$ with a load-balancing bias $\mathbf{b}$ updated per step with rate γ.

D.4 LTE [65]

Expert Grouping. LTE adopts the same expert construction strategy as D2DMoE, clustering FFN neurons in the parameter space to form balanced expert partitions [37]. For SwiGLU-based FFNs (e.g., LLaMA), LTE clusters the d_h gate neurons, i.e., the columns of the gate matrix W_g, and obtains an equal-size partition $\{\mathcal{C}_e\}_{e=1}^{K}$ with $|\mathcal{C}_e| = d_h/K$. This partition then induces the corresponding expert parameter blocks $\{W_g^{(e)}, W_{\text{up}}^{(e)}, W_{\text{down}}^{(e)}\}$.

Router and Adaptive Selection. LTE adopts a single-layer router that maps the FFN input x to an expert score for each expert, using independent Sigmoid scoring:

$$G(x)_e = \sigma(xW_{g,e}) = \frac{1}{1 + e^{-xW_{g,e}}}, \tag{30}$$

where $W_{g,e}$ denotes the router weight for expert e. At inference, experts are selected by thresholding:

$$FFN(x) = \sum_{e=1}^{n} \mathbf{1}_{\{G(x)_e > \tau\}} \, E_e(x), \tag{31}$$

with a fixed threshold $\tau = 0.5$ across all layers, as in LTE.

Two-stage Training. LTE employs a two-stage training scheme. Stage 1 jointly optimizes the router and model parameters under a soft routing relaxation,

$$FFN_{\text{soft}}(x) = \sum_{e=1}^{n} G(x)_e \, E_e(x), \tag{32}$$

together with the task loss and two regularizers (efficiency and separability) that encourage sparse and threshold-separable expert scores:

$$\mathcal{L}_{s1} = \mathcal{L}_{\text{task}} + \eta \, \mathcal{L}_{\text{eff}} + \lambda \, \mathcal{L}_{\text{sep}}. \tag{33}$$

Stage 2 freezes the router, switches to discrete threshold routing, and fine-tunes only model parameters with $\mathcal{L}_{\text{task}}$ to adapt to hard expert selection.

E Overall Algorithmic Workflow of ExpertNavigator

F Impact Statement

This paper presents ExpertNavigator, a four-stage Dense-to-MoE conversion pipeline that improves inference efficiency via structured sparse activation while maintaining high fidelity to the original

Algorithm 1 ExpertNavigator

Require: Dense model $\mathcal{M}$ with SwiGLU FFNs; data subsets $\mathcal{D}_{\mathrm{grp}}, \mathcal{D}_{\mathrm{rt}}, \mathcal{D}_{\mathrm{alloc}}, \mathcal{D}_{\mathrm{ste}}$; experts per layer K; global budget $\bar{k}$; temperatures T, τ; #pairs P.

Ensure: Converted model $\mathcal{M}_{\mathrm{MoE}}$ with experts $\{E_e^{(\ell)}\}$, routers $\{R^{(\ell)}\}$, and budgets $\{k_\ell\}$.

Phase I: Functionally Coherent Expert Grouping

for $\ell = 1$ to L **do**
  Define neuron function $f_j(x) = \big(\mathrm{SiLU}(xW_{g,j}) \odot (xW_{\mathrm{up},j})\big)W_{\mathrm{down},j}$ and contribution $c_j(x) = \|f_j(x)\|_2$.
  Estimate affinity $S_{ij}^{(\ell)} = \mathbb{E}_{x\sim\mathcal{D}_{\mathrm{grp}}}[c_i(x)c_j(x)]$.
  Partition neurons into equal-size clusters $\{\mathcal{C}_e^{(\ell)}\}_{e=1}^{K}$ by greedy equal-size graph partitioning on $S^{(\ell)}$.
  Slice $(W_g^{(\ell)}, W_{\mathrm{up}}^{(\ell)}, W_{\mathrm{down}}^{(\ell)})$ into experts $\{E_e^{(\ell)}\}$ according to $\{\mathcal{C}_e^{(\ell)}\}$.
end for

Phase II: Pairwise-Ranked Router Training

Initialize routers $\{R^{(\ell)}\}$; freeze expert weights.
for batch $x \sim \mathcal{D}_{\mathrm{rt}}$ **do**
  for $\ell = 1$ to L **do**
    Compute teacher scores $s_e^{(\ell)} = \|E_e^{(\ell)}(x)\|_2$, logits $\mathbf{z}^{(\ell)} = R^{(\ell)}(x)$.
    Sample $\mathcal{P}^{(\ell)} \subseteq \{(i,j) \mid s_i^{(\ell)} > s_j^{(\ell)}\}$ with $|\mathcal{P}^{(\ell)}| = P$.
  end for
  $\mathcal{L}_{\mathrm{rank}} = \frac{1}{L}\sum_{\ell=1}^{L} \frac{1}{|\mathcal{P}^{(\ell)}|} \sum_{(i,j)\in\mathcal{P}^{(\ell)}} \log\big(1 + \exp(-(z_i^{(\ell)} - z_j^{(\ell)}))\big)$.
  Update $\{R^{(\ell)}\}$ by descending $\nabla\mathcal{L}_{\mathrm{rank}}$.
end for

Phase III: Adaptive Layer-wise Sparsity Allocation

for $\ell = 1$ to L **do**
  Calculate $\mathrm{CV}_\ell = \mathbb{E}_{x\sim\mathcal{D}_{\mathrm{alloc}}}\left[\frac{\mathrm{Std}(\mathbf{s}^{(\ell)}(x))}{\mathrm{Mean}(\mathbf{s}^{(\ell)}(x))+\epsilon}\right]$, $\quad w_\ell = \frac{1}{\mathrm{CV}_\ell+\epsilon}$.
end for
$p_\ell = \mathrm{softmax}(w_\ell/T)$; allocate layer-wise top-k budgets $\{k_\ell\}$ under the global activation budget according to $\{p_\ell\}$.

Phase IV: STE-based Joint Adaptation

for batches from $\mathcal{D}_{\mathrm{ste}}$ **do**
  for $\ell = 1$ to L **do**
    $\mathbf{z}^{(\ell)} = R^{(\ell)}(x)$; $\mathbf{m}^{(\ell)} = \mathrm{TopKMask}(\mathbf{z}^{(\ell)}, k_\ell)$; $\mathbf{p}^{(\ell)} = \mathrm{softmax}(\mathbf{z}^{(\ell)}/\tau)$.
    $\hat{\mathbf{m}}^{(\ell)} = \mathbf{m}^{(\ell)} + (\mathbf{p}^{(\ell)} - \mathrm{StopGrad}(\mathbf{p}^{(\ell)}))$.
    $\mathrm{FFN}_{\mathrm{MoE}}^{(\ell)}(x) = \sum_{e=1}^{K} \hat{m}_e^{(\ell)}\, E_e^{(\ell)}(x)$ (*forward hard, backward soft*).
  end for
  Optimize all parameters with LM loss.
end for

dense model. By reducing compute and latency requirements, our approach can lower deployment cost and energy consumption, potentially broadening access to capable language models. Our method does not add new capabilities beyond those of the underlying pretrained model, and responsible use should follow established safety, evaluation, and access-control practices.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims are clearly presented in the abstract and introduction, and are further elaborated and substantiated throughout the subsequent sections of the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Refer to Sec. 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: [N/A]

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Refer to Sec. 4.1 and Appendix. B.1.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We use publicly available open-source data for all experiments, and provide sufficient implementation and training details in the paper and supplemental material to reproduce the main results. The project page is available at https://github.com/jianke0604/ExpertNavigator.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: Refer to Sec. 4.1 and Appendix. B.1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Following common practice in large-scale LLM studies, we do not report error bars or repeated runs. We use fixed training and evaluation settings, and assess robustness across multiple base models, activation budgets, benchmarks, and ablation settings.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: All experiments are conducted using 8 NVIDIA H100 GPUs. We provide the main training and evaluation settings in the paper and supplemental material, including data usage, activation budgets, training tokens, and evaluation benchmarks, to support reproducibility.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We have reviewed and are aware of the NeurIPS Code of Ethics, and confirm that the research conducted in this paper conforms to it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Refer to Sec. F.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.

- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The creators and original owners of all assets used in the paper are properly credited, and the licenses and terms of use are explicitly mentioned and fully respected.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.