HookMoE: A learnable performance compensation strategy of Mixture-of-Experts for LLM inference acceleration

Anonymous ACL submission

Abstract

Mixture of Experts (MoE) architectures have emerged as a promising paradigm for scaling model capacity through top-k routing mechanisms. Although reducing the number of activated experts inherently enables inference acceleration, this efficiency gain typically comes at the cost of significant performance degradation. To address this trade-off between efficiency and performance, we propose *HookMoE*, a plug-and-play single-layer compensation framework that effectively restores performance using only a small post-training calibration set. Our method strategically inserts a lightweight trainable Hook module immediately preceding selected transformer blocks. Comprehensive evaluations on four popular MoE models, with an average performance degradation of only 2.5% across various benchmarks, our method reduces the number of activated experts by more than 50% and achieves a 1.42× inference speed-up during the prefill stage. Through systematic analysis, we further reveal that the upper layers require fewer active experts, offering actionable insights for refining dynamic expert selection strategies and enhancing the overall efficiency of MoE models. [1]

1 Introduction

Large language models (LLMs), built on the Transformer architecture (Vaswani, 2017), have substantially reshaped the landscape of natural language processing by enabling advanced language understanding and generation across various applications (Brown, 2020; Achiam et al., 2023). Traditional dense models involve all parameters in computation, which can lead to inefficiencies, especially as the size of the models increases (Kaplan et al., 2020). In contrast, Mixture of Experts (MoE) models have emerged as a key variant of LLMs, addressing challenges of scalability and computational efficiency (Lepikhin et al., 2020; Fedus et al., 2022), and a key distinguishing feature of MoE models are active parameters. For example, Mixtral-8x22B (Jiang et al., 2024) only activates 39B parameters, allowing it to perform on a par with LLaMA3-70B on multiple evaluation tasks (Dubey et al., 2024).

Despite these advantages, deploying large-scale MoE models in real-world applications remains challenging due to high inference costs. As model sizes grow and user demands for complex reasoning increase, the need for efficient inference has become increasingly critical in both academia and industry. Current reasoning strategies such as Chain of Thought (CoT) (Wei et al., 2022) require a model to process longer token sequences at inference time. Recent progress in high-bandwidth memory[2] lets modern GPUs keep all expert parameters on the chip, so weights no longer need to be offloaded to the CPU. With this memory bottleneck removed, researchers can now focus on improving computational efficiency during inference.

MoE routes each token to the top-k experts, excluding all nonactivated experts from the computation. This activation sparsity makes it possible to reduce the number of active parameters even further. Recent studies reduce the running time cost by dynamically decreasing the number of experts per token (Huang et al., 2024; Szatkowski et al., 2023), but aggressive reduction often leads to significant performance degradation.

In this paper, we introduce a systematic compensation strategy that maintains model quality while keeping only a minimal set of experts active. Typically, we adjust the top-k routing to limit the number of activated experts and introduce a lightweight, learnable **Hook** compensation module to mitigate performance degradation. This component can be seamlessly integrated into existing

[1] The code will be released upon acceptance.

[2] See https://www.nvidia.com/en-us/data-center/h200 for details of the NVIDIA H200.

MoE architectures. We demonstrate the efficacy of our approach by applying it to four popular MoE models and evaluating their performance across multiple benchmarks. Remarkably, our approach strikes an advantageous balance with negligible extra training effort: merely 0.14% to 0.70% of the overall parameters need fine-tuning on a calibration dataset comprising only 512 samples. Moreover, each baseline model can complete its training in four hours when using 8 H800 GPUs. Compared to top-k routing alone, Hook compensation reduces the number of activated experts by 50%–75%. This decrease in expert activation directly yields computational savings, delivering over a 1.42× inference speed-up in the prefill stage.

Moreover, we analyze layer-depth based dynamic-k gating and observe that post-trained MoE models usually can activate fewer experts in their upper layers, and introduce the strategy Top Layer Needs Less Experts (**TLNLE**). Combined with TLNLE, **HookMoE** outperforms existing State of the Art (SOTA) dynamic-k methods by over 0.6% in overall performance while reducing the same number of activated experts. Our contributions are threefold:

Lightweight Hook Compensation Module. We propose a lightweight Hook compensation method that inserts a pluggable module before the Feed-Forward layer in a single transformer block. This module learns state-space corrections on a calibration dataset to counteract the performance loss from reduced expert usage, only increasing tiny parameters for fine-tuning.

Layer-Depth Based Dynamic-k Gating. We systematically vary the number of active experts between layers and discover the TLNLE pattern. TLNLE identifies layers, typically near the middle, that are most sensitive to expert count and are therefore chosen for hook insertion. Meanwhile, it also refines the current SOTA dynamic-k gating and further cuts the computation without sacrificing accuracy.

Comprehensive Quantitative and Qualitative Experiments. We conducted thorough evaluations of our method on four widely used MoE models, covering the parameters from 14.3 billion to 140.6 billion, and measured the inference speed-up on two different NVIDIA GPUs. Both quantitative metrics and qualitative analyzes show that our approach improves accuracy while activating fewer experts than competitive baselines.

2 Related Works

2.1 Mixture-of-Experts Models

Mixture-of-Experts (MoEs), first introduced by Jacobs et al. (1991), employ a gating mechanism to route inputs to a subset of experts, typically selecting the top-k most relevant experts per token. MoEs can be categorized by expert size: coarse-grained expert models like Mixtral (Jiang et al., 2024) deploy fewer but larger general-purpose experts, whereas fine-grained expert models such as Qwen (Bai et al., 2023) and DeepSpeed-MoE (Dai et al., 2024) feature many smaller, specialized experts, offering enhanced flexibility and specialization without incurring excessive overhead.

2.2 Dynamic-k Gating

Conventional top-k gating in MoEs assumes the same number of experts is activated for each token, neglecting the variability in task complexity across different inputs and potentially resulting in redundant computation. Thus, recent studies have revisited the core mechanism of expert activation under the framework of Dynamic-k Gating. For example, NAEE (Lu et al., 2024) selects experts based on predefined ratios inferred from importance scores, effectively eliminating less critical experts. D2D (Huang et al., 2024) adopts a probabilistic approach, dynamically adjusting expert activation according to input complexity. Furthermore, MoED (Szatkowski et al., 2023) leverages the sparsity of activation to determine the optimal number of experts per input in real time.

2.3 Parameter-Efficient Fine-Tuning

Parameter-Efficient Fine-Tuning (PEFT) provides a practical means of adapting LLMs to downstream tasks without incurring excessive computational or storage overhead (Han et al., 2024). Typically, these strategies can be categorized into three approaches: introducing additional parameters by freezing most of the base model and fine-tuning a small set of newly added parameters (Houlsby et al., 2019); selecting existing parameters by limiting the scope of fine-tuning to a restricted subset of the original parameters (He et al., 2023); and applying low-rank adaptations, such as LoRA (Hu et al., 2021), which decompose the weight matrices of the model into low-dimensional components. In this study, PEFT are employed to mitigate the performance degradation that arises when the number of active experts is reduced.

3 Method

In this section, we first present the preliminary MoE, and then introduce the proposed compensation module for the MoE model, with the aim of minimizing performance loss due to reduced activated experts. We also provide corresponding mathematical derivation.

3.1 Preliminary

The general architecture of the MoE layer consists of N experts $\{e_1, e_2, \ldots, e_N\}$, and each expert is a neural network capable of processing input tokens. For each input token $\boldsymbol{x}_i$ in the sequence, the router calculates a set of routing weights as follows:

$$P(\boldsymbol{x}_i) = \text{Softmax}(\boldsymbol{x}_i \cdot W), \tag{1}$$

where $W \in \mathbb{R}^{h \times N}$ is a learned projection matrix and $\boldsymbol{x}_i$ is the embedding of the token, h is the dimension of hidden states. The routing weights determine the relevance of each expert for the given input token. In practice, the top-k experts with the highest routing weights are selected, with k being a hyperparameter of the model, and these experts then process the input token.

The top-k routing strategy ensures that only a small subset of experts contribute to each token's output, which significantly reduces the computational cost. The output of the MoE layer for the token $\boldsymbol{x}_i$ is a sum of the output of the selected experts. Thus, the output for the token can be expressed as:

$$\boldsymbol{z}_i = \frac{\sum_{j \in \text{top-}k} P_j \cdot e_j(\boldsymbol{x}_i)}{\sum_{m \in \text{top-}k} P_m} \tag{2}$$

where $e_j(\boldsymbol{x}_i)$ is the output of the j-th activated expert, and P_j is the routing weight of the j-th expert.

In the case where the MoE utilizes shared experts, the output $\boldsymbol{z}_i$ is modified to incorporate the outputs of all shared experts in addition to the top-k selected experts. The output of the token $\boldsymbol{x}_i$ is then the sum of the outputs of both selected and shared experts:

$$\boldsymbol{z}_i = \boldsymbol{z}_i + \sum_{s \in \text{Shared}} e_s(\boldsymbol{x}_i). \tag{3}$$

where $e_s(\boldsymbol{x}_i)$ represents the output of the s-th shared expert.

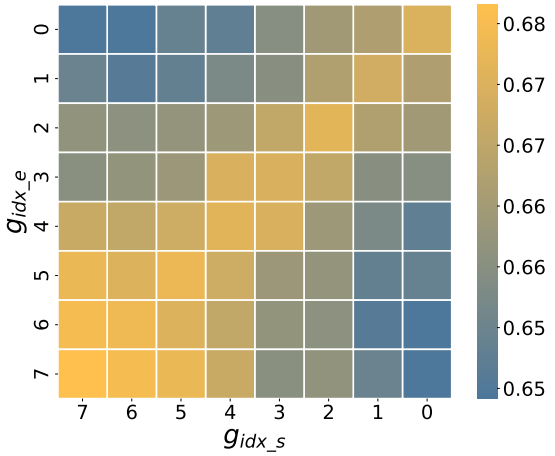

Figure 1: Mixtral-8x7B consists of 32 layers, which are structured into 8 groups, with each group containing 4 layers, with group boundaries defined by start index g_{idx_s} and end index g_{idx_e}. For layers within the range $[4 \times g_{idx\_s}, 4 \times g_{idx\_e} + 3]$, we apply the function $Expert\_N(k, l, r)$ with fixed $k = 2$, $l = 4 \times g_{\text{idx}_s}$, and $r = 4 \times g_{\text{idx}_e} + 3$. The effectiveness of this configuration is evaluated across general tasks, with average performance scores visualized and detailed results presented in Tab. 6.

3.2 Layer-Depth based Dynamic-k Gating

Inspired by work on overthinking in neural networks (Kaya et al., 2019; Huang et al., 2024), we conduct a simple study to measure how dynamically varying the top-k gate size as a function of layer depth affects a post-trained MoE model. We hypothesize that activating fewer experts in deeper layers within a layer range $[l, r]$ has a limited impact on model performance. Consequently, we define the number of active experts in the layer L as follows, where k is the activate expert number of base model:

$$Expert\_N(k, l, r) = \begin{cases} k - 1, & l \leq L \leq r, \\ k, & \text{otherwise.} \end{cases} \tag{4}$$

We systematically sweep the boundaries l and r and evaluate general-purpose tasks using the benchmarks described in Sec.4.1. The experiments were performed on Mixtral-8x7B. The results in Fig.1 reveal a clear pattern that we term Top Layers Need Less Experts (TLNLE). This insight enables a decrease in the number of active experts, while causing only a slight decline in performance. Specifically, when l is 16 and r is 31, which means layers $[16, 31]$ activate only the top-1 expert, while retaining top-2 experts for layers from 0 to 15, we observe only a 1.1% decrease in performance.

Based on this observation, we find that the right

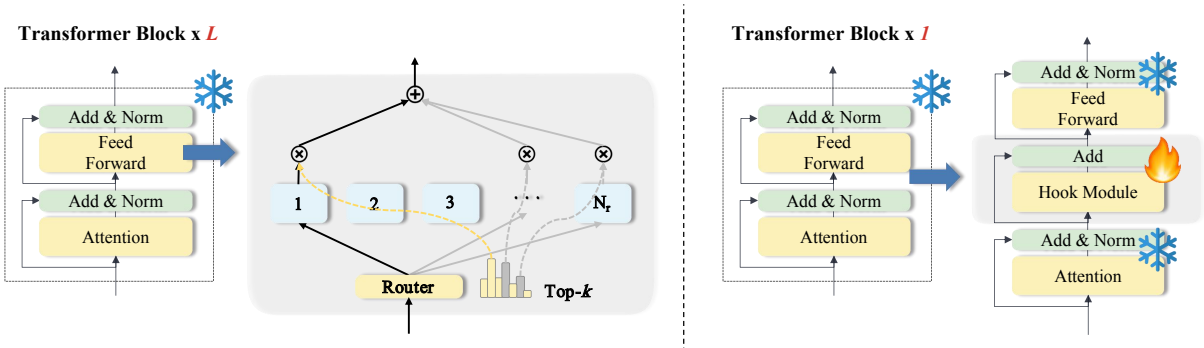

Figure 2: The two-stage framework of our proposed Hook compensation method. (a) Stage 1 employs the TLNLE strategy to reduce the number of activated experts and identifies a suitable layer for Hook Module integration. (b) Stage 2 incorporates the Hook Module, consisting of two linear layers with ReLU activation, before the Feed-Forward component of the selected layer. Detailed architecture is presented in Sec.3.3.

boundary r can be fixed to the total number of layers without loss of generality. Therefore, TLNLE can be parameterized solely by the left boundary l. Formally, we define:

$$TLNLE(k, l) = Expert\_N(k, l, r_{\max}), \quad (5)$$

where $r_{\max}$ denotes the index of the final layer in the model. This strategy effectively reduces computational overhead by limiting the number of experts utilized in the top layers, thereby accelerating inference, while still leveraging the full potential of the MoE for complex tasks.

3.3 A Hook Compensation Module

In this section, we present the proposed Hook compensation method, a fine-tuning approach that introduces an additional learnable module into the selected layer of the model, guided by the TLNLE strategy, to mitigate the performance loss caused by reducing the number of activated experts in MoE models. The strategy identifies the most suitable layer for inserting the Hook module by evaluating the degree of performance degradation between different candidate layers. This evaluation is based on the average scores on general tasks, as introduced in Sec. 4.1.

Specifically, we apply TLNLE starts by initializing k as the base number of routed experts per layer and l as the number of model layers. We first fix l, and iteratively reduce k until either (i) the performance drop exceeds a predefined threshold p, or (ii) $k = 2$, since at least one expert must remain active in each MoE layer to preserve functionality.

The parameter p controls the trade-off between model efficiency and performance stability. A smaller p results in fewer reductions and better performance preservation, while a larger p allows more aggressive pruning at the cost of potential performance loss. In our experiments, we typically set $p = 1.0\%$, which provides a reasonable balance between efficiency gains and model quality.

After determining the optimal k, we proceed to reduce l iteratively using the same performance deviation criterion based on p. Finally, we further reduce the number of activated experts by setting all layers to use $k - 1$ experts, and insert the Hook module into the selected layer l, as illustrated in Fig. 2. This fine-tuning phase aims to compensate for performance degradation caused by the additional reduction in expert activation before the layer l.

The Hook Module consists of two lightweight and trainable linear layers, which are added as a residual connection to the original input x_i. The module processes x_i and compensates for potential activation loss as follows:

$$x_i = x_i + \text{Linear}(\text{ReLU}(\text{Linear}(x_i))) \times r \quad (6)$$

Here, r denotes a scaling factor, which is a training parameter initialized to 0.5. The latent space size of the Hook module for coarse-grained experts is set to the intermediate size of the MoE, whereas for fine-grained experts in MoE, the latent space size is twice the MoE intermediate size. The output of Eq. 6 is then used as the input of Eq. 1.

To better understand how the Hook module mitigates the impact of expert reduction, we analyze the behavior of the model from a state space perspective. In this context, state space refers to the representation space formed by intermediate acti-

vations across the model layers, reflecting how information is transformed through the network. We define the state space vector of the original MoE as $\mathbf{S}_o$, the state space of the MoE with reduced activated experts as $\mathbf{S}_c$, and the state space of the Hook module as $\mathbf{S}_H$. The objective is to ensure that $\mathbf{S}_H$, the state space of the learnable module, effectively compensates for the loss caused by the reduced number of activated experts. This is achieved by minimizing the discrepancy as follows:

$$\min(\|\mathbf{S}_H - \mathbf{S}_c + \mathbf{S}_o\|) \tag{7}$$

Next, applying the triangle inequality to the L2 norm of the state space difference, we expand the left-hand side as follows:

$$\|\mathbf{S}_H - \mathbf{S}_c + \mathbf{S}_o\|_2^2 \leq \|\mathbf{S}_H - \mathbf{S}_c\|_2^2 + \|\mathbf{S}_o\|_2^2 \tag{8}$$

Direct evaluation of the model state space before and after compression is not feasible. Therefore, we rely on an evaluation dataset that defines the state space represented by the model on the validation set as $\mathbf{S}_v$. This allows us to further expand $\|\mathbf{S}_H - \mathbf{S}_c\|_2^2$:

$$\begin{aligned}\|\mathbf{S}_H - \mathbf{S}_c\|_2^2 &= \|\mathbf{S}_H - \mathbf{S}_v + \mathbf{S}_v - \mathbf{S}_c\|_2^2 \\ &\leq \|\mathbf{S}_H - \mathbf{S}_v\|_2^2 + \|\mathbf{S}_v - \mathbf{S}_c\|_2^2 \end{aligned} \tag{9}$$

Combining the above, we seek to minimize the following:

$$\min\left(\|\mathbf{S}_H - \mathbf{S}_v\|_2^2 + \|\mathbf{S}_v - \mathbf{S}_c\|_2^2 + \|\mathbf{S}_o\|_2^2\right) \tag{10}$$

Since $\mathbf{S}_c$, $\mathbf{S}_v$, and $\mathbf{S}_o$ are considered fixed, our primary objective is to minimize $\|\mathbf{S}_H - \mathbf{S}_v\|_2^2$. Given that the training data of the MoEs are inaccessible, we introduce a small calibration data set to compensate for activation losses as much as possible. From the expression, it is clear that to improve the efficacy of this method, the state space of the calibration dataset should closely match that of the evaluation dataset. Furthermore, selecting a diverse evaluation dataset ensures that the data distribution of the evaluation set is more representative of real-world scenarios. The calibration data set is introduced in Sec. 4.1.

4 Experiment

In this section, we first present an overview of the model, the calibration datasets, and the benchmarks in Sec. 4.1. We then systematically evaluate our proposed HookMoE framework from three primary perspectives. Specifically, in Sec. 4.2, we describe the experiments on the Hook module for both general and domain-specific tasks. In Section 4.3, we illustrate how the TLNLE strategy enhances the dynamic routing method and the Hook module. Finally, in Sec. 4.4, we assess the effectiveness of our approach in acceleration of inference.

Model	Expert Count (Activ. / Total)	$\mathbf{d}_{\text{expert}}$	Share Expert Count	Attention
Mixtral-8x7B	2 / 8	d_{ffn}	0	GQA
Mixtral-8x22B	2 / 8	d_{ffn}	0	GQA
Qwen1.5-14.3B-A2.7B	8 / 64	$\frac{1}{4}d_{\text{ffn}}$	4	MHA
DeepSeek-V2-Lite	8 / 66	$\approx \frac{1}{8}d_{\text{ffn}}$	2	MLA

Table 1: Architectural specifications of the benchmark models

4.1 Implementation Details

Model Settings. A comprehensive systematic evaluation was conducted to validate the effectiveness of the proposed framework in four state-of-the-art MoE architectures. Mixtral-8×7B (Jiang et al., 2024), Mixtral-8×22B(Jiang et al., 2024), DeepSeek-V2-Lite (Dai et al., 2024), and Qwen1.5-MoE-A2.7B (Bai et al., 2023). The architecture details are shown in Tab. 1, these models demonstrate distinct architectural implementations in both expert routing strategies and attention mechanisms. Specifically, the Mixtral series adopts a uniform top-k routing mechanism across all expert modules, integrated with Grouped-Query Attention (GQA) (Ainslie et al., 2023) for efficient computation. In contrast, Qwen1.5-MoE-A2.7B and DeepSeek-V2-Lite implement a hybrid expert management system, where shared experts remain permanently activated while routed experts are dynamically selected, combined with Multi-Head Attention (MHA) (Vaswani, 2017) and Multi-Head Latent Attention (MLA) (Dai et al., 2024) mechanisms, respectively.

Calibration Datasets and Benchmarks. Our calibration framework draws on two complementary corpora: the open domain C4 (Raffel et al., 2020) and the mathematically focused MATH (Hendrycks et al., 2021b). From each data set, we randomly sub-sample 512 examples for calibration. The C4 provides broad linguistic coverage, which is essential for cross-domain generalization, while the MATH specifically targets mathematical reasoning through its problem-solution pairs. Evaluation employs a series of standardized metrics to assess both linguistic and mathematical reasoning abilities, ensuring a comprehensive evaluation across diverse domains.

Method	Activated Params	General Tasks	Domain-Specific Tasks
None (Top-2)	12.88B	67.80	48.34
Top-1	**7.24B**	62.87	41.43
Ours (C4)	7.36B	**65.52**	40.16
Ours (Math)	7.36B	65.36	**41.91**

Table 2: Analysis of the effects of different calibration datasets on general and domain-specific tasks.

Evaluation employs a multi-aspect benchmark architecture comprising ten standardized tasks from lm-evaluation-harness (Gao et al., 2023). For general language understanding assessment under zero-shot settings, we utilize eight established benchmarks: ARC Challenge and ARC Easy (Clark et al., 2018), BoolQ and RTE (Wang et al., 2019), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2021a), OpenBookQA (Mihaylov et al., 2018), and Winogrande (Sakaguchi et al., 2019). Mathematical reasoning evaluation employs two specialized benchmarks: GSM8K (Cobbe et al., 2021) and Minerva Math (Hendrycks et al., 2021c).

Evaluation Metrics. For the evaluations, we first measure the model performance on the benchmarks mentioned above. Subsequently, we evaluate the model's inference acceleration using the Hook strategy. In particular, we track four key metrics: average activated expert counts per token (Act), average scores on evaluated tasks (Lm-Eval), time to first token (TTFT) and inter-token latency (ITL). Following NVIDIA's recommended LLM inference metrics[3], TTFT is calculated as the latency between the arrival of a request and the generation of the first token during the pre-fill stage, while ITL is computed as the latency between successive token generations for the same request in the decoding stage.

4.2 Compensation Performance for Models

In this subsection, we evaluate the proposed Hook method in both general and domain-specific tasks to provide a comprehensive assessment of model performance when reducing the number of activated experts.

Experiment Setup. We conduct experiments on four widely used MoE models, setting the performance degradation tolerance r to 1.5%. Using a binary search procedure, we determine the specific layers for inserting the Hook module. For each selected layer, the Hook module is placed in front of the gate within the MoE module, and the top-k parameter is adjusted accordingly.

For general tasks, we used 512 calibration samples from the C4 dataset, each with a sequence length of 2048. For domain-specific tasks, we sample 512 instances from the MATH dataset, also with a sequence length of 2048. For Mixtral-8x7B and Mixtral-8x22B, the hidden dimension of Hook module is set to the intermediate size of the expert, whereas for Qwen1.5-14.3B-A2.7B and DeepSeek-V2-Lite (fine-grained expert models), the hidden dimension is configured to be twice the intermediate size of the expert. All parameters remain frozen during fine-tuning, except those within the Hook module, which constitute only 0.14%–0.70% of the total parameters. We set the learning rate to 1×10^{-4} and fine-tuned it for 1000 epochs.

Once the Hook module is fine-tuned, we evaluate the performance of the compressed MoE models. For general tasks, we report zero-shot accuracies on eight benchmarks. For domain-specific math tasks, we provide 4-shot results for Minerva Math and 5-shot results for GSM8K. Due to space constraints, we present results for domain-specific tasks only for Mixtral-8x7B.

Compensation performance for General Tasks. Tab. 3 summarizes the precision, parameter count, and number of activated experts for the four baseline models. We also provide baseline results under a lower top-k activation for reference. Across all four models, our Hook method delivers superior performance compared to the original model, despite activating fewer experts.

Concretely, activating only one expert in Mixtral-8x7B reduces activated parameters to 57.1% compared to two activated experts, at the cost of a mere 2.5% performance drop. Using the Hook module for accuracy compensation, performance improves by an additional 2.43% compared to simply reducing the number of active experts without Hook. Similarly, activating a single expert in Mixtral-8x22B reduces the activated parameters to 57.3%, with a performance decrease of 1.68%, and Hook recovers 3.01%. For Qwen1.5-14.3B-A2.7B, activating an expert yields 76.9% of the original parameters activated (with four experts), causing only a performance degradation of 1.87%, while Hook further recovers 0.31%. Lastly, activating three experts in DeepSeek-V2-Lite reduces the activated parameters to 76.4% of the original six-expert setting, reducing performance by 1.81%, which Hook mitigates by 0.32%. In general, Hook activates only 25%–50% of the experts compared

[3]Our implementation is based on https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html.

Model	Activated Params	Total Params	Act.	ARC-c	ARC-e	BoolQ	HellaSwag	MMLU	OBQA	RTE	WinoGrande	Average
Mixtral-8x7B												
Origin (k=2)	12.90B	46.70B	2.00	56.66	84.34	85.23	64.82	67.88	35.20	71.48	76.80	67.80
Top-k (k=1)	7.24B	46.70B	1.00	50.90	78.87	80.80	60.65	61.60	31.40	68.23	70.48	62.87
Hook	7.36B	46.82B	**1.00**	55.80	82.07	85.63	64.06	62.82	34.40	64.26	73.32	**65.30** (+2.43)
Mixtral-8x22B												
Origin (k=2)	39.15B	140.60B	2.00	59.39	85.94	88.07	67.15	74.35	37.00	69.31	80.58	70.22
Top-k (k=1)	22.24B	140.60B	1.00	52.99	81.78	84.01	63.24	68.86	34.00	63.54	75.85	65.53
Hook	22.44B	140.80B	**1.00**	58.19	84.97	85.29	66.38	69.28	34.80	72.90	76.48	**68.54** (+3.01)
Qwen1.5-14.3B-A2.7B												
Origin (k=4)	2.69B	14.31B	4.00	46.08	75.34	86.42	62.84	74.12	33.20	74.37	74.74	65.89
Top-k (k=1)	2.06B	14.31B	1.00	46.76	77.06	85.26	57.49	66.45	30.20	75.09	71.35	63.71
Hook	2.07B	14.32B	**1.00**	48.38	77.48	85.35	58.10	66.34	30.00	76.17	70.32	**64.02** (+0.31)
DeepSeek-V2-Lite												
Origin (k=6)	2.80B	16.40B	6.00	46.59	78.32	79.08	58.55	54.81	33.40	61.37	70.32	60.31
Top-k (k=3)	2.13B	16.40B	3.00	42.92	77.02	76.09	56.73	52.00	32.00	60.29	68.35	58.18
Hook	2.14B	16.41B	**3.00**	43.77	77.06	76.18	56.81	51.80	31.80	62.45	68.11	**58.50** (+0.32)

Table 3: Performance comparison of four general MoE models under various top-k settings with and without the proposed Hook method. For each model, we report the number of activated parameters, total parameters, and the zero-shot accuracies on eight general tasks, followed by the overall average.

to the original models, recovering an additional 0.31%–3.01% accuracy relative to simply lowering the top-k without Hook.

Compensation performance for Domain-Specific Tasks. As shown in Tab. 2, for domain-specific tasks, using task-specific calibration data yields better performance than using the general pre-trained dataset. Hence, when selecting calibration data, it is advantageous to use data that closely align with the target domain.

Analysis of Performance Gains Across Models. Hook module consistently achieves strong performance across different MoE architectures, with the most pronounced gains observed in coarse-grained models, where it significantly outperforms the baseline on all benchmarks. In fine-grained MoE models, such as Qwen1.5-14.3B-A2.7B and DeepSeek-V2-Lite, it still achieves statistically significant improvements on five of the same benchmarks, albeit with smaller absolute gains. In such architectures, individual experts are less capable and there is greater redundancy among them. Consequently, reducing activation sparsity results in only a slight performance drop, limiting the potential for compensation. However, consistent outperformance of the baseline on the majority tasks demonstrates that Hook module is broadly applicable and effective on diverse granularities of MoE.

4.3 Effect of TLNLE

In this subsection, we explore the broader applicability of the TLNLE strategy, including its integration with existing dynamic-k methods and the Hook module. The results are based on the Mixtral-8x7B model.

Experiment Setup. In the experiment combining our method with the SOTA dynamic-k gating methods, we replicate the baselines of dynamic-k gating and incorporate the TLNLE strategy. The dynamic-k gating method uses a threshold to determine whether to skip an expert. The TLNLE strategy adjusts this threshold layer by layer, reducing the number of activated experts in top layers. Specifically, in the MoED (Szatkowski et al., 2023)experiment, the threshold p starts at 0.7 for the first layer and decreases by 0.2/32 for each subsequent layer. In the D2D (Huang et al., 2024) experiment, the threshold τ starts at 0.5 for the first layer and increases by 0.2/32 for each subsequent layer. In the Hook experiment, TLNLE activates two experts in the two bottom layers and one expert in the remaining layers. We report the Act and LM-Eval metrics. The Act results are derived from approximately 1 billion tokens in the router's choice statistics.

Route	Act.	Lm-Eval
Origin (Top-2)	2.00	67.80
MoED ($p = 0.6$)	1.31	66.37
MoED ($p = 0.7$)	1.59	67.06
D2D ($\tau = 0.5$)	1.50	67.17
D2D ($\tau = 0.6$)	1.38	66.68
NAEE	1.50	67.32
MoED + TLNLE	1.33	**66.82**
D2D + TLNLE	1.39	**67.19**
MoED ($p = 0.52$)	1.07	64.88
D2D ($\tau = 0.9$)	1.09	64.96
Hook + TLNLE	1.06	**65.52**

Table 4: Performance comparison of different routing methods.

Baselines for Comparison. Previous studies, such as MoED (Szatkowski et al., 2023), D2D (Huang et al., 2024), and NAEE (Lu et al.,

2024), have implemented dynamic-k routing using fixed thresholds for expert activation. In these methods, expert activation is determined by comparing the router's output to predefined thresholds.

Evaluation Results. As shown in Tab.4, compared to the baselines, the integration of our method with MoED and D2D results in minimal increases in the Act. MoED + TLNLE activates only 0.02 more experts than MoED ($p = 0.6$), yet achieves a 0.45 increase in accuracy. Similarly, D2D + TLNLE activates 0.01 more experts than D2D ($\tau = 0.6$), but outperforms D2D ($\tau = 0.5$). Compared to the SOTA dynamic-k gating method, the combination of Hook + TLNLE achieves an improvement of over 0.6% in overall performance, with a similar number of activated experts.[4]

4.4 Acceleration of Inference Speed

In this subsection, we apply our proposed method to the Mixtral-8x7B model and evaluate its performance on two key inference metrics. We conducted experiments on two different types of GPU to assess the effectiveness of our method in accelerating the inference speed.

Experiment Setup. The experiments are carried out using `transformers` Python library. For the TTFT measurement, the input sequence lengths are set at 1024, 2048, and 4096 tokens, and the output length is set at 1 token. Timestamps are recorded immediately before and after inference, with each measurement repeated 100 times. For ITL measurement, the setup remains the same, except that the maximum number of new tokens is set to 256. The experiments are carried out on systems with 1 node featuring 2x NVIDIA A800-SXM-80GB GPUs (400GB/s NVLink bandwidth and Bfloat16 Tensor Core with 312 TFLOPS) and 1 node with 2x NVIDIA H20-SXM-96GB GPUs (900GB/s NVLink bandwidth and Bfloat16 Tensor Core with 148 TFLOPS). The A800 is a compute-oriented GPU, known for its high single-card performance, while the H20 is an inference-oriented GPU, optimized for high inter-GPU communication bandwidth. Model inference is performed using the Bfloat16 representation.

Evaluation Results. As illustrated in Fig.3, our method achieves a TTFT speedup ranging from 1.24x to 1.46x, and an ITL speedup between 1.09x and 1.13x, compared to the baseline model. In particular, the speed-up is more pronounced on the inference-oriented H20 GPU than on the compute-oriented A800 GPU. This outcome aligns with our goal of accelerating inference for post-trained models while preserving accuracy. Our approach is particularly well-suited for deployment on inference-optimized GPUs.

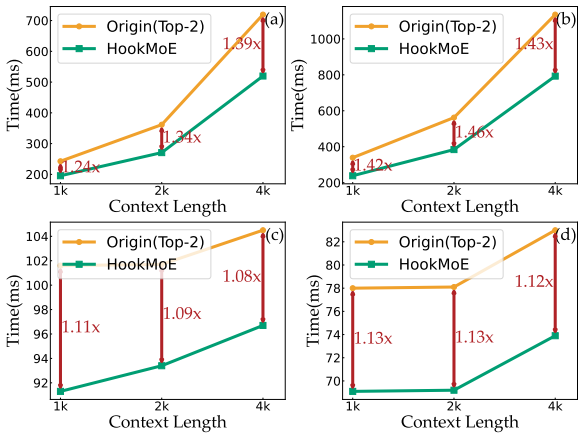

Figure 3: Performance comparison of TTFT and ITL metrics on Mixtral-8x7B model. The subfigures demonstrate: (a) TTFT measurements using A800 GPU; (b) TTFT measurements using H20 GPU; (c) ITL measurements using A800 GPU; (d) ITL measurements using H20 GPU.

By activating fewer experts, our method directly lowering the dominant GPU compute cost. However, as revealed by profiling with **Nvidia Nsight**, CPU scheduling overhead alone can account for up to 50% of the total inference time (Srivatsa et al.). Thus, although our current implementation achieves a TTFT speedup ranging from 1.24x to 1.46x under these conditions, we anticipate that this speedup will become increasingly as modern inference engines, such as vLLM (Kwon et al., 2023) and SGLang (Zheng et al., 2024), continue to improve their CPU scheduling efficiency.

5 Conclusion

In this paper, we propose a novel and lightweight Hook compensation module for MoE models, effectively addresses the challenge of balancing computational efficiency and model performance. Our approach introduces minimal training overhead and demonstrates consistent effectiveness across various MoE models and hardware platforms. By integrating this module with the observed TLNLE property, we offer a practical solution for deploying efficient yet performant MoE models. This work not only enhances the understanding of MoE architectures but also provides valuable insights for future research on efficient LLMs deployment.

[4]Futher details of Act are provided in A.4.

Limitations

Our approach effectively reduces the number of experts activated during inference and compensates for performance degradation by adding parameters, making it more efficient to deploy the mixture of experts (MoE). Despite the improvements, there are several limitations. First, the hook method demonstrates stronger compensation when fewer experts are activated. However, as the number of activated experts increases and approaches the original setting, the accuracy compensation may become less significant. Second, due to resource constraints, we have not evaluated MoEs with over 141B parameters, such as DeepSeek-V3, Grok-1 and Qwen3. As MoE continue to evolve and computational resources expand, we plan to conduct experiments on larger models to more comprehensively assess the generalizability and scalability of our method.

Ethics Statement

Our research focuses on mitigating performance degradation caused by reducing the number of activated experts by adding parameters for fine-tuning, with the goal of improving inference speed without sacrificing model performance. Although our approach offers potential benefits for more efficient deployment of advanced large language models (LLMs), we acknowledge the importance of carefully considering the ethical implications of deploying such models. The responsible use of LLMs involves addressing biases, improving the interpretability of the output, protecting data privacy, and conducting risk assessments. We are committed to making our code transparent for the responsible review and evaluation of the research community.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. 2023. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *ArXiv*, abs/1803.05457.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Preprint*, arXiv:2110.14168.

Damai Dai, Chengqi Deng, Chenggang Zhao, RX Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y Wu, et al. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

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

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. 2023. A framework for few-shot language model evaluation.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. Parameter-efficient fine-tuning for large models: A comprehensive survey. *arXiv preprint arXiv:2403.14608*.

Haoyu He, Jianfei Cai, Jing Zhang, Dacheng Tao, and Bohan Zhuang. 2023. Sensitivity-aware visual parameter-efficient fine-tuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11825–11835.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021c. Measuring mathematical problem solving with the math dataset. *NeurIPS*.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Quzhe Huang, Zhenwei An, Nan Zhuang, Mingxu Tao, Chen Zhang, Yang Jin, Kun Xu, Liwei Chen, Songfang Huang, and Yansong Feng. 2024. Harder tasks need more experts: Dynamic routing in moe models. *arXiv preprint arXiv:2403.07652*.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

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

Yigitcan Kaya, Sanghyun Hong, and Tudor Dumitras. 2019. Shallow-deep networks: Understanding and mitigating network overthinking. In *International conference on machine learning*, pages 3301–3310. PMLR.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*.

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

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *arXiv preprint arXiv:1907.10641*.

Vikranth Srivatsa, Dongming Li, Yiying Zhang, and Reyna Abhyankar. Can Scheduling Overhead Dominate LLM Inference Performance? A Study of CPU Scheduling Overhead on Two Popular LLM Inference Systems. https://mlsys.wuklab.io/posts/scheduling_overhead/.

Filip Szatkowski, Bartosz Wójcik, Mikołaj Piórczyński, and Kamil Adamczewski. 2023. Sadmoe: Exploiting activation sparsity with dynamic-k gating. *arXiv e-prints*, pages arXiv–2310.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

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

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. 2024. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*.

A Appendix

A.1 Evaluation of Adjust Top-k

In the top-k adjustment experiment, we control the number of experts activated per layer by modifying the number of activated experts parameter in the configuration file, varying it from 1 to 8. Fig. 4 illustrates the results of varying top-k settings. As observed, performance is optimal with the original top-2 configuration, while performance significantly drops with top-1. However, when increasing the number of experts from the top-3 to top-8, performance still declines slightly. This phenomenon shows that after the key experts are activated, the model's ability to resist noise increases.

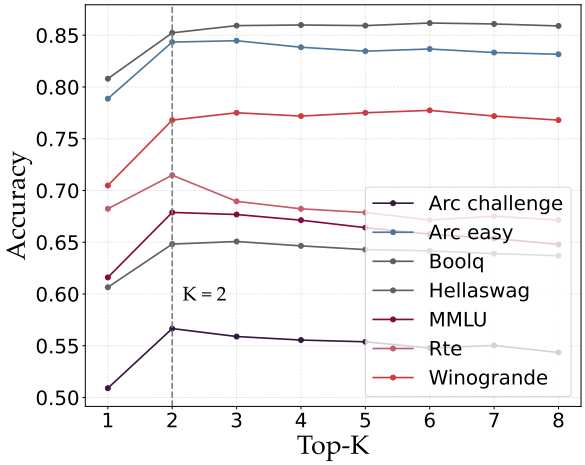

Figure 4: Evaluation Mixtral-8×7B in different general tasks.

A.2 Parameter Calculate for MoE Models

Consider a Mixture of Experts (MoE) model with the following parameters:

$$P = L \times (P_{\text{Attention}} + P_{\text{MoE}}) + P_{\text{Embedding}} + P_{\text{LM\_head}}$$

Let L be the number of transformer blocks, h represent the hidden size, $h_{ffn-expert}$ the hidden size of the expert, k the number of experts selected per token, n_e^r the total number of router experts, n_e^s the total number of shared experts, and v the size of the vocabulary. The total number of parameters in the model is given by:

$$\mathcal{P}_t = L \times \left(h^2 \times 4 + h \times h_{ffn\_moe} \times 3 \times (n_e^r + n_e^s)\right) + 2 \times v \times h$$

The term $h^2 \times 4$ corresponds to the Attention layer parameters $P_{\text{Attention}}$, and the term $h \times h_{ffn\_moe} \times 3 \times (n_e^r + n_e^s)$ accounts for the MoE layers P_{MoE}, including both routing and shared experts. Both parameters $P_{\text{Embedding}}$ and $P_{\text{LM\_head}}$ are $v \times h$. Similarly, the activation values are calculated as follows:

$$\mathcal{P}_a = L \times \left(h^2 \times 4 + h \times h_{ffn\_moe} \times 3 \times (k + n_e^s)\right) + 2 \times v \times h$$

This expression captures the contributions from both the standard transformer layers and the MoE layers, as well as the embedding layer corresponding to the vocabulary.

A.3 Training Curve

In this subsection, we present the loss curve for four baseline models during training in Fig. 5. A consistent trend is observed across all four MoE models: a steady reduction in loss. This indicates that the models effectively explored and implemented more optimal activation compensation strategies.

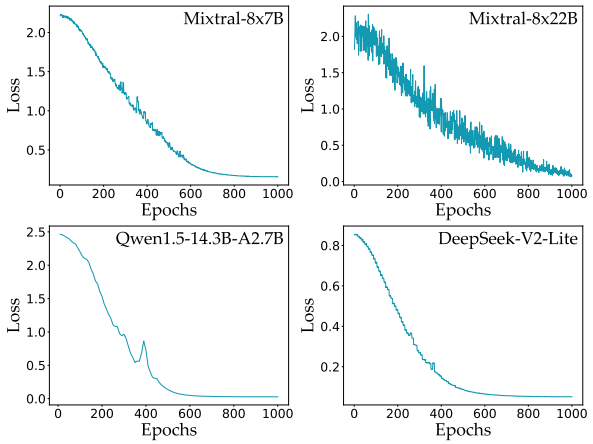

Figure 5: Loss curves for four baseline models.

A.4 Details of Dynamic-k Routing

In this subsection, we present a comprehensive analysis of the average number of activated experts at each of the 32 layers in Mixtral-8×7B, evaluated over 2.1M tokens. Our investigation compares various dynamic-k routing strategies to highlight their impact on the utilization of experts across different layers. Details are shown on Tab.5.

A.5 LLM Inference Metrics

Time to First Token (TTFT): This metric measures the latency that a user experiences before receiving the model's output. TTFT covers the prefill time. Specifically, TTFT increases with the length

Layer	MoED	D2D	TLNLE
0	1.04	1.05	2.00
1	1.07	1.08	2.00
2	1.06	1.08	1.00
3	1.07	1.09	1.00
4	1.06	1.08	1.00
5	1.07	1.08	1.00
6	1.08	1.10	1.00
7	1.08	1.10	1.00
8	1.09	1.11	1.00
9	1.09	1.11	1.00
10	1.09	1.11	1.00
11	1.06	1.08	1.00
12	1.07	1.09	1.00
13	1.10	1.13	1.00
14	1.09	1.11	1.00
15	1.08	1.10	1.00
16	1.09	1.11	1.00
17	1.07	1.09	1.00
18	1.08	1.09	1.00
19	1.07	1.08	1.00
20	1.07	1.08	1.00
21	1.07	1.09	1.00
22	1.07	1.09	1.00
23	1.07	1.09	1.00
24	1.07	1.08	1.00
25	1.06	1.08	1.00
26	1.06	1.08	1.00
27	1.06	1.07	1.00
28	1.05	1.07	1.00
29	1.06	1.07	1.00
30	1.05	1.07	1.00
31	1.08	1.10	1.00
Act.	1.07	1.09	1.06
LM-Eval	64.88	64.96	65.52

Table 5: Layer-wise comparison of the average number of activated experts for MoED ($p = 0.52$), D2D ($\tau = 0.9$), and TLNLE (2,2).

of the input prompt due to the attention mechanism's requirement to process the entire input sequence and build the key-value cache (KV cache), which is essential for initiating the iterative token generation loop. The longer the prompt, the greater the TTFT, as more time is needed to prepare the KV-cache.

Inter-token Latency (ITL): ITL represents the average time between the generation of consecutive tokens, also known as the time per output token (TPOT). ITL is formally defined as:

$$\text{ITL} = \frac{\text{e2e\_latency} - \text{TTFT}}{\text{Total\_output\_tokens} - 1}$$

Here, e2e_latency denotes the total end-to-end latency of the inference process, and TTFT is the time to the first token. The subtraction of 1 from the denominator reflects the fact that the first token generation is excluded from the ITL calculation, as it pertains to the prefill stage, not the iterative decoding process.

It is important to note that as the length of the output sequence increases, the size of the KV-cache also grows, which increases memory consumption. Additionally, the computational cost of the attention mechanism increases linearly with the length of the input and output sequence generated so far. Although this computation is typically not compute bound, the growing memory demand can impact overall performance.

A.6 Activation Visualization

In this subsection, we visualize the activation values during Mixtral-8×7B inference to qualitatively assess the impact of our method on the output. Specifically, we input the word HooKMoE into the tokenizer, which divides the word into three tokens: Hook, M, and OE. These tokens are then sequentially fed into the model for forward propagation. We extract the tensor at the last index from the final layer, resulting in a tensor of size 4096. The extracted tensor is then reshaped to a 64x64 matrix, normalized, and visualized. The results are shown in Fig.6. As illustrated, although our method activates only the top-1 expert, the activation compensation through the Hook Module ensures that the visualized output at the bottom layer closely approximates that of the original model.

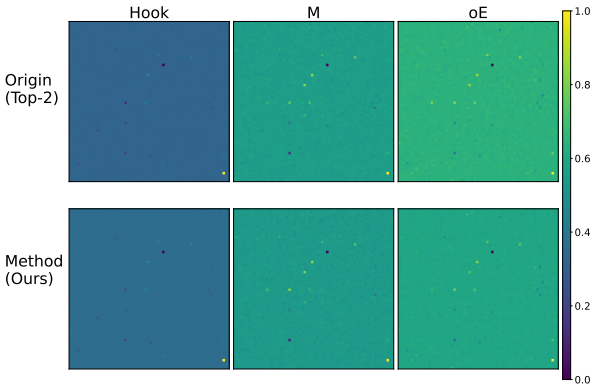

Figure 6: Visualization of activation values at the bottom layer during model inference with three tokens.

A.7 Benchmark Descriptions

The following section provides a brief overview of the benchmarks used in our evaluation. These benchmarks are designed to assess various aspects of language understanding and reasoning capabilities across diverse domains. Tab. 7 summarizes each benchmark and its corresponding task description.

g_{idx_s}	g_{idx_e}	Act	ARC-c	ARC-e	BoolQ	HellaSwag	MMLU	OBQA	RTE	WinoGrande	Average
0	0	1.88	57.34	83.92	85.41	64.76	67.25	33.40	69.68	74.11	66.98
0	1	1.75	56.23	83.38	85.11	64.27	66.43	33.20	66.79	74.19	66.20
0	5	1.25	55.12	81.99	85.32	62.78	63.47	32.00	64.98	73.40	64.88
0	6	1.13	54.35	81.31	85.38	62.50	63.47	31.40	64.62	72.30	64.42
1	2	1.75	55.12	83.29	85.17	64.05	65.23	33.80	67.51	75.61	66.22
1	3	1.63	55.03	82.49	85.02	63.83	63.84	34.00	66.06	73.72	65.50
1	4	1.50	53.67	82.41	85.26	63.34	63.82	33.80	64.98	74.74	65.25
1	5	1.38	53.07	81.65	84.98	62.81	63.90	33.00	64.26	74.90	64.82
1	6	1.25	52.90	81.14	85.02	62.85	63.92	32.00	65.34	73.72	64.61
1	7	1.13	53.84	81.19	85.20	62.77	63.85	32.40	66.06	74.27	64.95
3	3	1.88	55.89	83.63	85.60	64.64	66.18	34.20	69.31	76.24	66.96
3	4	1.75	55.89	83.63	85.60	64.64	66.18	34.20	69.31	76.24	66.96
4	5	1.75	54.95	82.41	85.17	63.82	67.83	33.80	70.40	75.61	66.75
4	6	1.63	54.52	82.49	84.83	63.52	67.87	32.00	71.48	75.45	66.52
4	7	1.50	55.12	82.11	84.83	63.56	67.79	33.20	70.76	75.93	66.66
6	6	1.88	56.57	84.09	85.38	64.63	68.02	33.40	70.40	76.40	67.36
0	2	1.63	55.29	83.33	85.14	63.62	65.00	32.80	67.87	74.43	65.94
0	3	1.50	54.78	82.79	84.98	63.63	63.54	34.60	65.70	73.72	65.47
0	4	1.38	54.78	82.74	85.17	63.05	63.15	32.60	64.26	72.61	64.80
0	7	1.00	54.35	81.31	85.38	62.50	63.47	31.40	64.62	72.30	64.42
1	1	1.88	55.46	83.46	85.54	64.56	66.68	33.60	68.59	76.40	66.79
2	2	1.88	56.40	84.18	85.11	64.37	66.42	35.20	69.68	75.77	67.14
2	3	1.75	55.03	83.21	85.32	64.18	64.85	35.00	68.23	76.32	66.52
2	4	1.63	53.92	82.74	85.41	63.72	64.78	34.00	67.15	75.30	65.88
2	5	1.50	53.67	81.90	85.47	63.10	65.12	33.80	67.51	75.14	65.71
2	6	1.38	53.58	81.48	85.47	63.08	65.03	34.20	67.15	74.66	65.58
2	7	1.25	52.82	81.82	85.47	63.04	64.92	34.20	68.23	74.74	65.66
3	5	1.63	54.35	81.82	85.44	63.37	66.06	33.40	67.15	75.22	65.85
3	6	1.50	53.84	81.78	85.54	63.25	66.13	32.40	67.51	75.06	65.69
3	7	1.38	53.92	81.65	85.29	63.25	66.08	33.00	66.79	74.35	65.54
4	4	1.88	55.46	83.25	85.23	64.41	67.75	34.60	70.40	75.85	67.12
5	5	1.88	56.14	83.38	84.98	64.43	68.02	34.60	71.12	75.77	67.31
5	6	1.75	55.89	83.08	84.86	64.13	68.00	33.80	70.40	76.01	67.02
5	7	1.63	55.72	82.95	84.92	64.03	68.03	35.20	71.12	76.24	67.28
6	7	1.75	55.55	83.42	85.32	64.54	67.81	36.20	71.12	75.77	67.47
7	7	1.88	56.74	83.75	85.29	64.84	67.73	36.20	70.76	75.93	67.66

Table 6: Evaluation results of act and 8 general tasks under different sets of g_{idx_s} and g_{idx_e}.

Benchmark	Description
ARC-c / ARC-e	Tasks involving complex reasoning over a diverse set of questions.
HellaSwag	Tasks to predict the ending of stories or scenarios, testing comprehension and creativity.
MMLU	Massive Multitask Language Understanding benchmark for broad domain language evaluation. Several variants are supported.
OBQA	Open-book question answering tasks that require external knowledge and reasoning.
RTE / BoolQ	A suite of challenging tasks designed to test a range of language understanding skills.
WinoGrande	A large-scale dataset for coreference resolution, inspired by the Winograd Schema Challenge.

Table 7: Overview of the benchmarks used in the evaluation, including their descriptions and target capabilities.