Input-Aware Expert Pruning for Efficient MoE Deployment

Anonymous ACL submission

Abstract

Mixture-of-Experts (MoE) models, a primary method for scaling parameters to boost performance in large language models (LLMs), require substantial memory when deployed in downstream systems. To mitigate this, existing methods often prune or compress parameters before inference to reduce memory usage. Yet, such static optimizations conflict with the MoE design philosophy: expert activation is input-dependent. To resolve this issue, we introduce in**P**ut-awa**R**e **E**xpert **P**runing (PREP), a method that dynamically identifies and retains only the most critical experts for each input, substantially lowering memory overhead while preserving model performance. Specifically, after derivation of expert importance, PREP deploys an input-dependent lightweight linear approximation of expert importance through efficient search in CPU. Incorporating a hardware-optimized mechanism of layer-by-layer loading of the experts, PREP achieves a minimal memory usage of **37.5%** compared with the base model. Experiments across diverse benchmarks demonstrate that our method outperforms prior compression techniques in accuracy while achieving the lowest inference latency. Code for reproducibility is available at `https://anonymous.4open.science/r/PREP-5375`.

1 Introduction

Mixture-of-Experts (MoE) architectures have revolutionized natural language processing (NLP) by enabling models to scale parameters efficiently while activating only a subset of experts per input, thus balancing performance and computational cost (Antoniak et al., 2025; Li et al., 2025a). For example, the DeepSeek v3 (Liu et al., 2024b) model leverages MoE to achieve state-of-the-art results with 671B parameters, yet activates only 37B parameters per token, demonstrating MoEs capacity for sparse computation. However, deploying MoE models remains challenging due to their substantial memory demands (Xie et al., 2024b). The 671B-parameter DeepSeek-R1 (Guo et al., 2025) model requires over 600GB of GPU memory for inference, even the half-precision Mixtral 8x7B (Jiang et al., 2024) model still demands significant resources, often exceeding consumer-grade GPU limits. These requirements hinder practical deployment despite MoEs architectural benefits (Eliseev and Mazur, 2023).

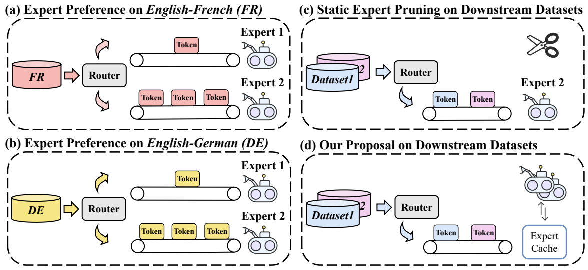

Figure 1: Illustration of expert preference across datasets and different pruning strategies. Subfigure (a) and (b) demonstrate the preferences of Expert 1 and Expert 2 for FR and DE datasets, respectively. Subfigure (c) illustrates the static pruning method for MoE models, while (d) presents the core idea of our proposal.

Previous work has explored optimization strategies to address these challenges, such as quantization (*e.g.,* assigning variable bit-widths to experts) (Li et al., 2024) and expert pruning (Xie et al., 2024a). These techniques, such as MC MoE (Huang et al., 2024) and Expert Sparsity (Lu et al., 2024), aim to compress experts by allocating lower bit-widths or discarding those considered less important, thereby reducing memory usage. In these methods, expert importance is typically assessed via metrics such as activation frequency or mean squared error between compressed and original outputs on calibration datasets like C4 (Raffel et al., 2020). However, these approaches often lack flexibility and incur significant computational costs (Xue et al., 2024; Gao et al., 2025). In particular, adaptive quantization necessitates retraining or re-quantizing models, while static pruning risks discarding experts critical for diverse inputs.

More critically, existing approaches often compress experts before deployment, overlooking the importance of input-dependent expert selection (Lv et al., 2025). Some studies on multilingual translation models reveal that expert utility varies significantly across languages (Li et al., 2023b). As illustrated in Figure 1, distinct experts exhibit pronounced performance preferences across different language translation datasets. This variability highlights a critical limitation: evaluating experts using static calibration data (*e.g.,* a fixed validation set) may degrade performance on downstream tasks with diverse or distribution-shifted inputs.

To address these limitations, we present PREP, the first training-free, input-aware expert pruning method for MoE models. PREP dynamically identifies critical experts per input during inference, retains them in GPU memory, and offloads redundant experts to system memory. This approach reduces peak GPU memory usage and accelerates inference speed without retraining. Specifically, PREP comprises two key procedures: **(1) Efficient Layer-Wise Expert Evaluation**: After quantifying expert's layer-wise influence, we propose an input-aware expert importance metric to measure each expert's contribution. We then approximate these importance scores using a linear method, effectively transforming the problem into finding the maximum value given an input query. Thereafter, we implement a fast CPU-based search to efficiently identify the most important experts. **(2) Adaptive Layer-wise Expert Loading**: Then, we analyze the importance of different layers and find that earlier (shallow) layers significantly influence the final decoding process. Consequently, we allocate a layer-specific number of experts to retain. Additionally, we employ a layer-wise loading strategy: the selected experts are loaded into memory, while the remaining ones are offloaded to system memory using a scheduled approach. This balances memory efficiency and inference latency. By combining input-aware evaluation and layer-wise loading, PREP optimizes MoE models for diverse inputs while maintaining performance. Overall, the main contribution of this work is:

- We propose an input-aware expert evaluation strategy that efficiently identifies the most important experts for a given input without requiring additional training.
- We introduce a hardware-friendly adaptive layer-wise loading, enabling efficient MoE deployment and achieving approximately 1.4× inference speedup compared to the original model.
- Extensive experiments conducted on multiple benchmark datasets reveal that the proposed PREP achieves state-of-the-art performance.

2 Related Works

2.1 Mixture-of-Experts in LLMs

The Mixture of Experts paradigm, which uses specialized sub-models (experts) and a gating mechanism to dynamically select and activate a subset of these experts based on the input, has become a key architecture for scaling large language models (LLMs) while maintaining computational efficiency. Building on classical sparse expert systems (Jacobs et al., 1991), modern MoE LLMs employ dynamic token-based expert routing, where each input selectively activates the top-k experts per layer through learned gating mechanisms (Shazeer et al., 2017; Lepikhin et al., 2020). This sparse activation reduces FLOPs by 60-70% compared to dense models with the same parameter count. The Switch Transformer (Fedus et al., 2022) was a pioneer in this area, replacing dense feed-forward layers with expert layers and achieving 7× faster pre-training than T5-Large. Subsequent work, Mixtral 8×7B (Jiang et al., 2024), extended this paradigm with a sparse decoder-only architecture, achieving LLaMA 2-70B level performance at 40% of the inference cost. The most recent breakthrough, DeepSeek-V3 (Liu et al., 2024b), leverages an innovative MoE architecture to match the performance of state-of-the-art proprietary models (e.g., GPT-4o and Claude-3.5-Sonnet) on human-aligned evaluation frameworks. Our work complements these innovations by focusing on memory-efficient inference, integrating seamlessly into existing MoE models without retraining. This orthogonal contribution addresses a critical gap in deployment scalability.

2.2 Expert Purning for MoE LLMs

While traditional MoE models activate only a subset of experts per input, their memory footprint remains large because all experts are stored in GPU memory (Zhao et al., 2025). To address this issue, recent work has explored two main strategies: static expert pruning and dynamic expert skipping. Static methods, such as EEP (Liu et al., 2024c) and MoE-I^2 (Yang et al., 2024), prune experts based on activation frequency from calibration datasets. However, these approaches ignore input-dependent variations in expert importance (Cheng et al., 2024; Lv et al., 2025). Dynamic methods like MoE++ (Jin et al., 2025) skip experts adaptively introducing zero-experts, but they provide only incremental computational gains without addressing the memory bottleneck caused by retaining all experts in GPU memory (Abnar et al., 2025). In this paper, we introduce an input-aware expert pruning strategy that dynamically offloads deactivated experts to system memory, thereby improving the efficiency of MoE deployment.

2.3 Expert Approximation for MoE LLMs

Expert approximation techniques aim to reduce computational costs by decomposing or adaptively assigning quantization bit-widths to experts while retaining all experts. Methods like MC-SMoE (Li et al., 2023a) and MoE-SVD (Li et al., 2025b) use Singular Value Decomposition (Abdi, 2007) for low-rank factorization of each experts parameters. Techniques such as BSP (Li et al., 2024) and MC-MoE (Huang et al., 2024) assign different bit-widths based on activation frequency and reconstruction loss from calibration datasets. However, these approaches ignore input-specific variations (Lv et al., 2025) and require re-quantizing and decomposing the entire model, which can be cumbersome (Sharma et al., 2025). Our proposed method only requires building a linear expert parameters index, enabling plug-and-play MoE deployment in downstream applications.

3 Preliminary

This section provides background on MoE architectures and formally defines the task of memory-efficient expert pruning.

MoE in LLMs In decoder-based MoE-LLMs with L transformer layers, the traditional Feed-Forward Network (FFN) is replaced with the MoE layer. Each MoE layer consists of K expert modules and a gating layer. Each expert module is a FFN with three linear layers separated by activation functions. Let $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ denote the input embedding matrix, where n is the token count and d is the feature dimension. Additionally, let $\boldsymbol{H}_l \in \mathbb{R}^{n \times d}$ represent the hidden state of the l-th MoE layer. The output of the l-th MoE layer is expressed as:

$$\boldsymbol{Y}_l = \sum_{k=1}^{K} [G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l). \quad (1)$$

Here, $[G_l(\boldsymbol{H}_l)]_k \in \mathbb{R}^n$ denotes the routing weights for the k-th expert across all tokens, and $E_{l,k}(\boldsymbol{H}_l) \in \mathbb{R}^{n \times d}$ is the output of k-th expert.

Memory Efficient MoE The goal of memory-efficient MoE LLMs is to retain a subset of experts to reduce memory usage while preserving the original next-token prediction distribution, denoted as p_{ori}. Specifically, given a retention threshold $\tau \in [0, K \times L]$, we aim to identify a subset of expert modules S with size $|S| = \tau$. This subset should ensure that the output distribution of the pruned model, p_S, closely matches the original distribution. Formally, this can be expressed as:

$$\arg\min_{|S|=\tau} \mathbb{E}_{\boldsymbol{X}\sim\mathcal{D}} \left[D_{\text{KL}} \left(p_{\text{ori}}(\boldsymbol{X}) \| p_S(\boldsymbol{X}) \right) \right],$$

where $\boldsymbol{X}$ is sampled from a specific data distribution $\mathcal{D}$, D_{KL} represents the KL divergence.

However, it poses two challenges: **1) Combinatorial complexity**: the search space grows combinatorially with $\binom{N \times L}{\tau}$, rendering exhaustive evaluation intractable. **2) Input-dependent dynamics**: expert importance varies dynamically with input patterns, causing the search space to scale with the dataset size. These issues motivate our proposed input-aware expert pruning approach, which dynamically selects the most critical experts per input, enabling efficient and adaptive pruning.

4 Input-Aware Expert Pruning

This section introduces the framework of PREP. Specifically, we first evaluate the layer-wise influence of experts by approximating the output variation induced by each expert. Based on this, we define the expert importance metric and linearize it, facilitating the efficient evaluation of each expert. Next, we analyze across-layer importance to assign layer-specific pruning thresholds, paired with a dynamic loading strategy to balance computational efficiency and memory usage. An overview of our method is illustrated in Figure 2.

4.1 Efficient Layer-wise Expert Evaluation

Layer-wise Expert Influence Analysis In MoE layers, the hidden state evolves dynamically across

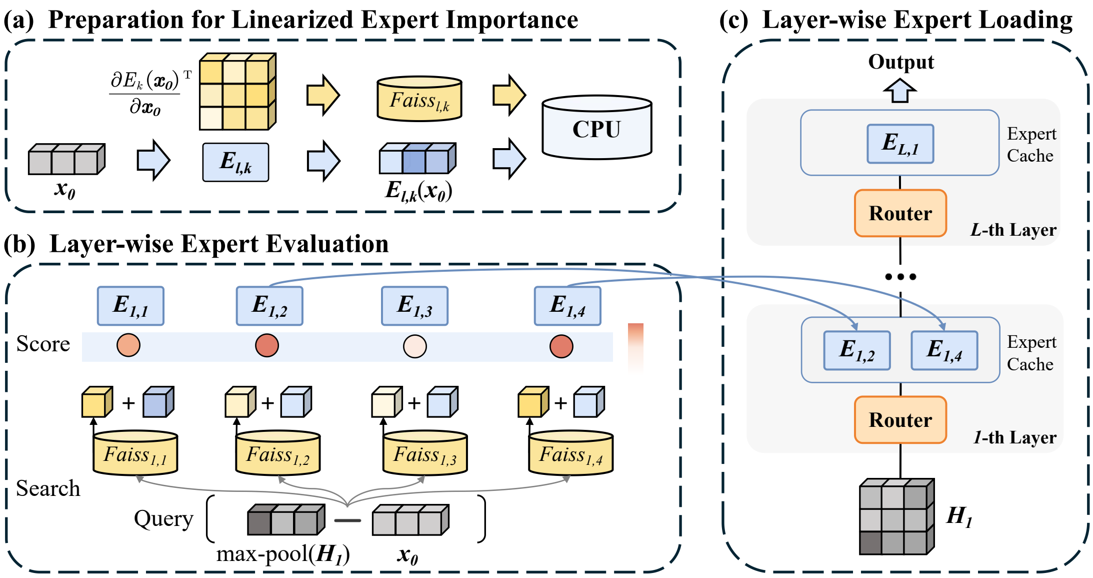

Figure 2: The overview of the proposed **PREP**. Subfigure (a) illustrates the process of collecting the output and Jacobian matrix of each expert at the expansion point during the offline stage. Subfigure (b) and (c) show each layer selects the most critical experts on the CPU for the given input and loads them into the Expert Cache.

layers, making it challenging to determine layer-specific pruning strategies based solely on input token embeddings. Instead, we select a subset of experts per layer to minimize output variation.

Formally, for the l-th MoE layer with retention threshold $\tau_l \in [0, K]$, we identify the optimal expert subset S_l to minimize the output variation $\Delta \boldsymbol{Y}_l$. Assume the pruned output of l-th layer as:

$$\hat{\boldsymbol{Y}}_l = \sum_{k \in S_l}^{|S_l|} [G'_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l), \tag{2}$$

where $G'_l(\boldsymbol{H}_l)_k$ denotes the adjusted routing weights after pruning. Then, the output variation $\Delta \boldsymbol{Y}_l = \boldsymbol{Y}_l - \hat{\boldsymbol{Y}}_l$ is decomposes into:

$$\begin{aligned}
\Delta \boldsymbol{Y}_l &= \boldsymbol{Y}_l - \hat{\boldsymbol{Y}}_l \\
&= \sum_{k \notin S_l}^{K-|S_l|} [G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l) \\
&+ \sum_{k \in S_l}^{|S_l|} ([G_l(\boldsymbol{H}_l)]_k - [G'_l(\boldsymbol{H}_l)]_k) \odot E_{l,k}(\boldsymbol{H}_l) \\
&\approx \sum_{k \notin S_l}^{K-|S_l|} [G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l).
\end{aligned} \tag{3}$$

The approximation in the last equation holds because the routing weights $[G_l(\boldsymbol{H}_l)]_k$ and $[G'_l(\boldsymbol{H}_l)]_k$ differ only in normalization, making their values similar for retained experts (More proof details are provided in Appendix A).

Approximated Expert Importance Following the derivation of Equation (3), we can upper bound the output variation (measured by its second norm) caused by pruning the k-th expert in the l-th layer (i.e., when $k \notin S_l$) as:

$$||[G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l)||_2 \leq ||[G_l(\boldsymbol{H}_l)]_k \cdot \sigma_{l,k} \cdot d||_2, \tag{4}$$

where $\sigma_{l,k} = \max_{i,j}[E_{l,k}(\boldsymbol{H}_l)]_{i,j}$ denotes the maximum activation value of the output of k-th expert in the l-th MoE layer across all n tokens, d represents the feature dimension of the hidden state. To minimize computational overhead, we assess the importance of the k-th expert in the l-th MoE layer using $\sigma_{l,k}$, which measures the maximum output perturbation caused by pruning the expert. However, directly evaluating this metric is computationally intensive for long sequences (large n), as it requires processing all n tokens.

To address this, we propose an efficient alternative: We first apply max-pooling to the input, extracting the maximum value in each feature dimension across all tokens, and then compute the final maximum value over the resulting vector. This reduces the computational complexity of evaluating each expert from $O(n)$ to $O(1)$, as follows:

$$\tilde{\sigma}_{l,k} = \max_j [E_{l,k}(\boldsymbol{h}_{l,\max})]_j, \tag{5}$$

where $\boldsymbol{h}_{l,\max} = \text{max-pool}(\boldsymbol{H}_l)$ is the max-pooling over $\boldsymbol{H}_l$ on feature dimension. This approach retains critical input features while enabling fast evaluation of expert importance based on a compressed input representation. We evaluate alternative expert evaluation strategies, *e.g.*, Predictor-based evaluation, in Section 5.3.1.

Linearized Expert Importance To enable efficient evaluation of expert importance without heavy GPU computation, we propose a linear approximation of expert importance using a first-order Taylor expansion. For an expert module $E_{l,k}$, let $\boldsymbol{x}_0 \in \mathbb{R}^d$ denote the expansion point. The approximated importance score $\tilde{\sigma}_{l,k}$ is derived as:

$$\tilde{\sigma}_{l,k} \approx \max_j \left[E_{l,k}(\boldsymbol{x}_0) + \left(\frac{\partial E_{l,k}(\boldsymbol{x})}{\partial \boldsymbol{x}} |_{\boldsymbol{x}=\boldsymbol{x}_0} \right)^\top (\boldsymbol{h}_{l,\max} - \boldsymbol{x}_0) \right]_j$$

where $E_{l,k}(\boldsymbol{x}_0)$ is the output of the k-th expert at the expansion point $\boldsymbol{x}_0$, and $\frac{\partial E_{l,k}(\boldsymbol{x}_0)}{\partial \boldsymbol{x}_0}^\top$ is the first-order gradient (*i.e.,* Jacobian matrix) at $\boldsymbol{x}_0$.

To determine the expansion point $\boldsymbol{x}_0$, we compute the mean of the hidden states from each MoE layer on the RedPajama dataset (Computer, 2023), using this value as the Taylor expansion point for each layer. We evaluate the different expansion points in Appendix D.

Fast Search of Important Expert Building on linearized expert importance, we propose a search-based method that replaces matrix multiplication for calculating expert importance on the CPU. In the indexing stage, we compute the Jacobian matrix $\frac{\partial E_{l,k}(\boldsymbol{x}_0)}{\partial \boldsymbol{x}_0}^\top$ for each expert at $\boldsymbol{x}_0$ and construct a Faiss index (Johnson et al., 2019). During inference, we treat $(\boldsymbol{h}_{l,\max} - \boldsymbol{x}_0)$ as a query vector and perform a rapid search of the Faiss index to retrieve the top-k maximum inner product and their corresponding indices. We use these indices to obtain the values of $E_{l,k}(\boldsymbol{x}_0)$ at the corresponding positions, add them to the retrieved results, and select the maximum value as the importance of expert.

4.2 Adaptive Layer-wise Expert Loading

A keen reader might wonder how to select τ_l for different layers. Therefore, this section further explores across-layer importance to determine the optimal τ_l and introduces layer-wise expert loading, enabling efficient dynamic expert pruning.

Across-Layer Importance Analysis Prior work has established that distinct layers in LLMs serve specialized functional roles (Fan et al., 2024; Zhang et al., 2024). To enable efficient layer-wise MoE deployment, we conduct an analysis to determine the importance of each MoE layer and assign layer-specific thresholds τ_l accordingly. Inspired by (Chuang et al., 2024), we propose an intuitive metric I_l for the l-th layer importance: the Jensen-Shannon divergence (Fuglede and Topsoe, 2004) between the model's original output distribution p_{ori} and its modified distribution p_l when excluding the l-th layer as:

$$I_l = \mathbb{E}_{\boldsymbol{X} \sim \mathcal{D}} \left[\frac{1}{2} D_{\text{KL}}(p_{\text{ori}}(\boldsymbol{X}) || p_m(\boldsymbol{X})) + \frac{1}{2} D_{KL}(p_l(\boldsymbol{X}) || p_m(\boldsymbol{X})) \right],$$

where $p_m = \frac{1}{2}(p_{\text{ori}} + p_l)$, $\boldsymbol{X}$ is sampled from a specific data distribution $\mathcal{D}$, D_{KL} represents the KL divergence.

Accordingly, we distribute the τ_l for different layers, guided by the layer importance distribution. The layer importance results and specific τ_l for different MoE LLMs are presented in Section 5.3.2.

Layer-wise Loading of Experts Previous studies have demonstrated that, with careful scheduling and design, **communication delays between the CPU and GPU do not become a bottleneck for inference latency** (Kwon et al., 2023; He et al., 2024). Drawing from this insight, we adopt a layer-wise expert loading strategy to implement input-aware expert pruning under limited computational resources. During decoding, each layer dynamically selects the most important experts based on the input, loading them into GPU memory while offloading others to system memory. This reduces GPU memory usage significantly. Additionally, we employ a fixed-size buffer per layer with a Least Recently Used (LRU) policy (Eliseev and Mazur, 2023) to retain recently active experts, minimizing delays from frequent swaps and enhancing inference efficiency.

5 Experiments

5.1 Experimental Settings

Dataset To evaluate the effectiveness of various expert compression methods, we conduct extensive experiments on several general benchmarks, including ARC-easy, ARC-challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag(Zellers et al., 2019), MMLU (Hendrycks et al., 2021) and Wino-Grande (Sakaguchi et al., 2021). The evaluation metrics and prompts follow the settings of Open-Compass (Contributors, 2023). Furthermore, we supplement our evaluation with perplexity analysis on the WikiText2 (Merity et al., 2016) and C4 (Raffel et al., 2020) datasets to assess the model's language modeling capability. The details of the datasets are reported in Appendix B.

Model	Params↓	Method	ARC-e	ARC-c	BoolQ	HellaSwag	MMLU	Winogrande	Average
Mixtral Instruct	25.0%	BSP	83.09	50.52	59.72	42.72	46.46	52.09	55.77
		EEP	86.94	73.67	83.43	63.99	49.25	60.24	69.59
		MC MoE	84.74	72.19	82.39	56.38	55.01	58.56	68.21
		Expert Sparsity	84.21	72.36	**85.35**	63.83	51.14	59.83	69.45
		PREP	**90.19**	**78.63**	80.85	**71.76**	**57.92**	**61.56**	**73.48**
	40.0%	EEP	81.36	68.62	72.04	58.23	43.15	55.36	63.13
		MC MoE	75.73	64.38	73.21	44.57	44.77	53.67	59.39
		Expert Sparsity	79.32	64.29	**79.94**	59.33	45.13	**58.09**	64.35
		PREP	**87.99**	**73.73**	77.55	**67.17**	**54.74**	57.70	**69.81**
Mixtral	25.0%	BSP	76.87	55.49	59.39	34.77	36.32	48.30	51.86
		EEP	79.53	64.27	71.84	42.62	50.41	54.23	60.48
		MC MoE	78.22	64.72	62.32	42.60	53.80	54.85	59.42
		Expert Sparsity	76.79	61.03	68.62	41.68	53.36	54.22	58.28
		PREP	**85.71**	**70.82**	**73.73**	**46.67**	**64.09**	**55.88**	**66.15**
	40.0%	EEP	70.36	60.64	65.04	34.62	43.52	51.75	54.32
		MC MoE	64.71	56.64	60.06	26.31	41.74	50.52	50.00
		Expert Sparsity	68.67	55.19	62.35	31.63	46.67	50.75	52.54
		PREP	**82.66**	**69.27**	**69.66**	**42.11**	**58.23**	**54.22**	**62.69**

Table 1: Zero-shot performance comparison of different methods under varying expert parameter compression ratios. "Params↓" represents the reduction radio in expert parameters. "Average" is calculated among six benchmarks. The best results are shown in **bold**.

Baselines We compare our method with the following two categories of expert compression:

- **Weight Quantization Methods** aim to retain all experts while assigning quantization weights based on each expert's importance. 1) **BSP** (Li et al., 2024) calculates each expert's importance score using a lightweight predictor to determine bit allocation. 2) **MC MoE** (Huang et al., 2024) formulates adaptive bit-width allocation as a linear programming problem, where the objective function balances multiple factors reflecting the importance of each expert.

- **Expert Pruning Methods** remove less important experts while applying the same quantization weight to all experts. 1) **Expert Sparsity** (Lu et al., 2024) enumerates all possible expert combinations at the layer level and retains the optimal combination by minimizing MSE loss. 2) **EEP** (Liu et al., 2024c) employs a gradient-free evolutionary search method, optimizing the pruning of experts within an efficient parameter space.

Experimental Protocols We employ Mixtral 8x7B and Mixtral 8x7B Instruct (Jiang et al., 2024) as our backbone for main evaluation. DeepSeek-V2-Lite-Chat (Liu et al., 2024a) serves as an additional backbone to further analyze the performance of our method, as detailed in Section 5.4. Both Mixtral 8x7B and Mixtral 8x7B Instruct consist of 32 transformer layers, each containing MoE blocks with 8 experts and employing a top-2 routing strategy. In contrast to traditional MoE architectures, DeepSeek-V2-Lite-Chat has 27 transformer layers, with the first layer utilizing a dense FNN, and the remaining layers incorporating MoE blocks. Each MoE block in DeepSeek-V2-Lite-Chat consists of two shared experts and 64 independent experts, using a top-6 routing strategy. To evaluate the model's performance, we use **Accuracy** and **Perplexity** as metrics. For inference speed, we measure **Latency**, defined as the time taken to generate each token. Additional experimental details are provided in Appendix C.3.

Model	Params↓	Method	Wiki	C4
Mixtral Instruct	25.0%	BSP	8.76	16.71
		EEP	5.82	8.99
		MC MoE	5.02	7.89
		Expert Sparsity	5.01	8.78
		PREP	**4.76**	**7.47**
	40.0%	EEP	6.37	13.79
		MC MoE	5.84	11.27
		Expert Sparsity	5.64	**9.35**
		PREP	**5.03**	11.02
Mixtral	25.0%	EEP	5.82	8.99
		BSP	6.43	17.40
		MC MoE	4.59	8.44
		Expert Sparsity	4.81	8.79
		PREP	**4.52**	**8.14**
	40.0%	EEP	7.37	15.79
		MC MoE	6.18	15.46
		Expert Sparsity	**5.62**	13.90
		PREP	5.72	**12.23**

Table 2: The perplexity for language modeling on WikiText2 and C4. The best results are shown in **bold**.

5.2 Main Results

This subsection presents models' performance on standard benchmarks (Table 1), their perplexity on language modeling (Table 2), and inference latency across varying input lengths (Figure 3). From the results, we have following observations: **1) PREP achieves the best performance, outperforming all baselines across all benchmarks.** Table 1 shows that PREP significant improvements over MC MoE and Expert Sparsity. These gains can be attributed to the reliance of MC MoE and Expert Sparsity on the C4 calibration dataset for pruning, which limits their generalization on downstream tasks. In contrast, EEP leverages task-specific pruning by directly optimizing configurations on subsets of each benchmark. However, this strategy yields suboptimal results and necessitates task-aware adjustments, rendering the method impractical for real-world deployment.

2) PREP achieves the best performance in natural language modeling, with the lowest perplexity across most datasets. We summarize perplexity results for various methods on WikiText2 and C4 in Table 2. Notably, on the C4 dataset, which shares distributional similarities with calibration data used in MC-MoE and Expert Sparsity, our approach outperforms others, highlighting the robustness of input-aware pruning under in-domain conditions. In contrast, EEP significantly underperforms, which is likely because EEP's pruning results are derived from narrow data subsets.

3) PREP exhibits the lowest latency when modeling inputs of varying lengths. As shown in Figure 3, our method delivers a 1.2× speedup over the fastest baselines (Expert Sparsity and EEP). This efficiency gain stems from two key innovations: 1) a carefully designed expert load strategy that minimizes redundant overhead, and 2) custom CUDA kernel optimizations that reduce inter-device communication latency. Finally, we evaluate the search time of PREP, which requires only 0.03 seconds per sample —a negligible cost accounting for less than 0.5% of total inference time.

5.3 Analysis Experiments

5.3.1 Validity of Expert Evaluation Strategy

This experiment evaluates the effectiveness of layer-wise expert evaluation strategy. In particular, we modify the proposed expert evaluation strategy with the following variants: **1)** Random Scoring: Experts are randomly scored for retention; **2)** Routing Weight-based evaluation: Experts are evaluated based on their cumulative routing weight contributions; **3)** Predictor-based evaluation: Experts are evaluated by a trained predictor (Details are provided in the Appendix C.2). We report the performance on the MMLU and HellaSwag benchmarks, with expert parameters reduced by 50.0%, 56.7%, and 67.7%, respectively, demonstrating the efficiency of our strategy.

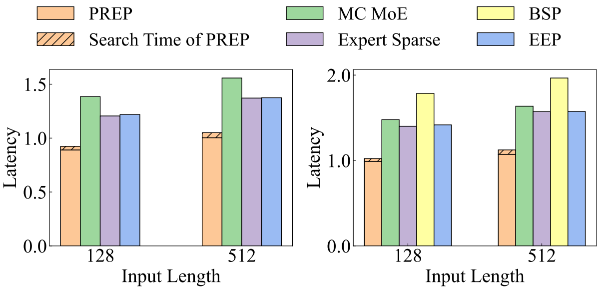

Figure 3: Inference speed comparison for different input lengths.

Params↓	Evaluation Strategy	MMLU	HellaSwag
50.0%	Random	40.14	47.51
	Routing weight	41.76	44.12
	Predictor	40.45	42.37
	Input-Aware (Ours)	**48.93**	**58.15**
56.7%	Random	37.13	41.09
	Routing weight	39.41	32.84
	Predictor	37.18	35.85
	Input-Aware (Ours)	**46.23**	**54.29**
67.7%	Random	34.41	32.34
	Routing weight	36.27	27.66
	Predictor	35.73	30.82
	Input-Aware (Ours)	**42.29**	**47.05**

Table 3: Performance comparison of different expert evaluation strategies for dynamic pruning. The best results are marked **bold**.

We summarize the results in Table 3. Our observations are as follows: **1) Determining the importance of each expert through routing weights and a trained predictor is challenging**, sometimes resulting in worse performance than Random Scoring. **2) Performance gaps between baseline methods and our strategy increases as parameter compression intensifies**, exposing fundamental limitations in these variants: their inability to dynamically identify input-critical experts and inherent robustness deficiencies.

5.3.2 Analysis of Across-Layer Importance

Following the definition in Section 4.2, we analyze the layer importance for Mixtral-8×7B Instruct and DeepSeek-V2-Lite-Chat on the RedPajama dataset, as shown in Figure 4. It can be observed

Model	Method	Params↓	ARC-e	ARC-c	BoolQ	HellaSwag	MMLU	Winogrande	Average
DeepSeek	Base	0	80.88	68.24	73.57	65.15	50.20	57.77	65.97
	PREP	12.5%	75.59	62.23	64.31	54.41	44.64	55.32	59.42
		25.0%	73.11	59.48	61.76	50.65	41.27	53.59	56.64

Table 4: Zero-shot performance comparison between Base and our method under varying expert parameter compression ratios, with Base serving as the upper limit.

Model	Method	Params↓	Latency_{128}	Latency_{512}
DeepSeek	Base	0	0.71	0.82
	PREP	12.5%	0.54	0.63
		25.0%	**0.49**	**0.61**

Table 5: Inference speed comparison for different input lengths. 'Latency_{128}' and 'Latency_{512}' represent the latency for input lengths of 128 and 512, respectively. The best results are marked **bold**.

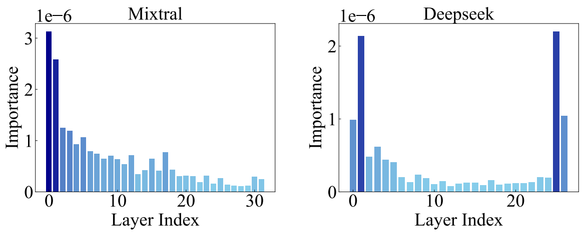

Figure 4: Layer importance for Mixtral-8×7B Instruct and DeepSeek-V2-Lite-Chat.

that both **models exhibit greater importance in the shallower layers.** Notably, DeepSeek-V2-Lite-Chat shows significantly higher importance in the last two layers, whereas the Mixtral-8×7B Instruct exhibits the opposite pattern. Based on the importance scores for each MoE layer on the calibration set, we assign layer-wise expert retention thresholds under a pre-defined expert parameter compression ratio. For the Mixtral model, we divide the 32 MoE layers into four groups. The number of experts retained between groups is distributed in a 2:2:1:1 ratio, with each group having an equal distribution of retention thresholds across its layers. For the DeepSeek model, we divide the first 24 MoE layers into four groups, with the remaining two layers assigned to a separate group. The number of experts retained between groups is allocated in a 2:2:1:1:1 ratio, with retention thresholds evenly distributed across the layers within each group.

5.4 More Analysis on DeepSeek

In this subsection, we apply our hardware-friendly input-aware pruning method to DeepSeek-V2-Lite-Chat to evaluate its generalizability. **It is important to note that existing pruning methods for MoE LLMs either do not support the DeepSeek model or have not released corresponding code.** For a baseline comparison, we use a uniform 4-bit quantization strategy for each expert and fully load the expert module via Layer-wise Loading of Experts, preserving lossless performance. In our experiments, we restrict each MoE block to retain at most one expert, prioritizing minimal memory usage.

Results are reported in Table 4 (zero-shot performance) and Table 5 (latency vs. input length). From the results, we have the following observations: **1) Our methods does not exhibit a significant performance decline compared to the base method.** Table 4 shows that pruning 12.5% and 25% of expert parameters reduces the models average performance by 6.55% and 9.33%, respectively. These declines are acceptable, as dynamic pruning on quantized models inherently introduces additional performance trade-offs. **2) The proposal effectively improves inference speed over the base method.** In Table 5, pruning 25% of expert parameters reduces latency by 30.14% for an input length of 128. These results demonstrate the method's effectiveness in balancing computational efficiency and memory constraints. Additionally, inference memory usage is approximately **4.5GB** under this setting.

6 Conclusion

This paper introduces an efficient input-aware expert pruning method, PREP, to tackle the memory and computational challenges in deploying MoE-based LLMs. By dynamically evaluating the linearized importance of each layer expert for a given input through a fast CPU-based search, and loading the most critical experts into GPU memory layer by layer with a well-designed scheduling strategy, PREP significantly reduces memory overhead and inference latency. Experimental results demonstrate that our approach outperforms the baseline on almost all benchmarks and achieves the lowest inference latency.

7 Limitations

Our proposed framework, PREP, significantly reduces memory usage and accelerates inference, making MoE-based LLMs more deployable. However, certain limitations persist. First, the input-aware expert pruning strategy does not support batch inference. This limitation arises because our method dynamically loads only the necessary experts based on each input's features, which complicates batch processing. Second, due to computational constraints, we have not evaluated PREP on larger MoE models such as Mixtral-8×22B (141B) and DeepSeek-V3 (671B). In future work, we intend to apply our method to these larger models to further assess its scalability and performance.

References

Hervé Abdi. 2007. Singular value decomposition (svd) and generalized singular value decomposition. *Encyclopedia of measurement and statistics*, 907(912):44.

Samira Abnar, Harshay Shah, Dan Busbridge, Alaaeldin Mohamed Elnouby Ali, Josh Susskind, and Vimal Thilak. 2025. Parameters vs flops: Scaling laws for optimal sparsity for mixture-of-experts language models. *arXiv preprint arXiv:2501.12370*.

Szymon Antoniak, Michał Krutul, Maciej Pióro, Jakub Krajewski, Jan Ludziejewski, Kamil Ciebiera, Krystian Król, Tomasz Odrzygóźdź, Marek Cygan, and Sebastian Jaszczur. 2025. Mixture of tokens: Continuous moe through cross-example aggregation. *Advances in Neural Information Processing Systems*, 37:103873–103896.

Hicham Badri and Appu Shaji. 2023. Half-quadratic quantization of large machine learning models.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. 2024. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. 2024. Dola: Decoding by contrasting layers improves factuality in large language models. In *The Twelfth International Conference on Learning Representations*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. BoolQ: Exploring the surprising difficulty of natural yes/no questions. 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 2924–2936, Minneapolis, Minnesota. Association for Computational Linguistics.

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 preprint arXiv:1803.05457*.

Together Computer. 2023. Redpajama: An open source recipe to reproduce llama training dataset.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation models. `https://github.com/open-compass/opencompass`.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library. *arXiv preprint arXiv:2401.08281*.

Artyom Eliseev and Denis Mazur. 2023. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, Yequan Wang, and Zhongyuan Wang. 2024. Not all layers of llms are necessary during inference. *arXiv preprint arXiv:2403.02181*.

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.

Bent Fuglede and Flemming Topsoe. 2004. Jensen-shannon divergence and hilbert space embedding. In *International symposium onInformation theory, 2004. ISIT 2004. Proceedings.*, page 31. IEEE.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Ying He, Jingcheng Fang, F Richard Yu, and Victor C Leung. 2024. Large language models (llms) inference offloading and resource allocation in cloud-edge computing: An active inference approach. *IEEE Transactions on Mobile Computing*.

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

Wei Huang, Yue Liao, Jianhui Liu, Ruifei He, Haoru Tan, Shiming Zhang, Hongsheng Li, Si Liu, and Xiaojuan Qi. 2024. Mc-moe: Mixture compressor for mixture-of-experts llms gains more. *arXiv preprint arXiv:2410.06270*.

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*.

Peng Jin, Bo Zhu, Li Yuan, and Shuicheng YAN. 2025. Moe++: Accelerating mixture-of-experts methods with zero-computation experts. In *The Thirteenth International Conference on Learning Representations*.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

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*.

Aonian Li, Bangwei Gong, Bo Yang, Boji Shan, Chang Liu, Cheng Zhu, Chunhao Zhang, Congchao Guo, Da Chen, Dong Li, et al. 2025a. Minimax-01: Scaling foundation models with lightning attention. *arXiv preprint arXiv:2501.08313*.

Pingzhi Li, Xiaolong Jin, Yu Cheng, and Tianlong Chen. 2024. Examining post-training quantization for mixture-of-experts: A benchmark. *arXiv preprint arXiv:2406.08155*.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. 2023a. Merge, then compress: Demystify efficient smoe with hints from its routing policy. *arXiv preprint arXiv:2310.01334*.

Shangjie Li, Xiangpeng Wei, Shaolin Zhu, Jun Xie, Baosong Yang, and Deyi Xiong. 2023b. MMNMT: Modularizing multilingual neural machine translation with flexibly assembled MoE and dense blocks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4978–4990, Singapore. Association for Computational Linguistics.

Wei Li, Lujun Li, You-Liang Huang, Mark G. Lee, Shengjie Sun, Wei Xue, and Yike Guo. 2025b. Structured mixture-of-experts LLMs compression via singular value decomposition.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. 2024a. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*.

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

Enshu Liu, Junyi Zhu, Zinan Lin, Xuefei Ning, Matthew B Blaschko, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. 2024c. Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs. *arXiv preprint arXiv:2407.00945*.

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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6159–6172, Bangkok, Thailand. Association for Computational Linguistics.

Ang Lv, Ruobing Xie, Yining Qian, Songhao Wu, Xingwu Sun, Zhanhui Kang, Di Wang, and Rui Yan. 2025. Autonomy-of-experts models. *arXiv preprint arXiv:2501.13074*.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

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. *J. Mach. Learn. Res.*, 21(1).

Ramin Raziperchikolaei and Young-joo Chung. 2024. One-class recommendation systems with the hinge pairwise distance loss and orthogonal representations. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pages 1033–1038.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99106.

Manish Sharma, Jamison Heard, Eli Saber, and Panos P Markopoulos. 2025. Convolutional neural network compression via dynamic parameter rank pruning. *IEEE Access*.

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

Thaddeus Tarpey. 2007. Linear transformations and the k-means clustering algorithm: applications to clustering curves. *the american statistician*, 61(1):34–40.

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

Yanyue Xie, Zhi Zhang, Ding Zhou, Cong Xie, Ziang Song, Xin Liu, Yanzhi Wang, Xue Lin, and An Xu. 2024a. Moe-pruner: Pruning mixture-of-experts large language model using the hints from its router. *arXiv preprint arXiv:2410.12013*.

Zhitian Xie, Yinger Zhang, Chenyi Zhuang, Qitao Shi, Zhining Liu, Jinjie Gu, and Guannan Zhang. 2024b. Mode: A mixture-of-experts model with mutual distillation among the experts. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(14):16067–16075.

Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. 2024. Moe-infinity: Activation-aware expert offloading for efficient moe serving. *arXiv preprint arXiv:2401.14361*.

Cheng Yang, Yang Sui, Jinqi Xiao, Lingyi Huang, Yu Gong, Yuanlin Duan, Wenqi Jia, Miao Yin, Yu Cheng, and Bo Yuan. 2024. MoE-I^2: Compressing mixture of experts models through inter-expert pruning and intra-expert low-rank decomposition. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10456–10466, Miami, Florida, USA. Association for Computational Linguistics.

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*, pages 4791–4800, Florence, Italy. Association for Computational Linguistics.

Jianyi Zhang, Da-Cheng Juan, Cyrus Rashtchian, Chun-Sung Ferng, Heinrich Jiang, and Yiran Chen. 2024. Sled: Self logits evolution decoding for improving factuality in large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 5188–5209. Curran Associates, Inc.

Changyuan Zhao, Hongyang Du, Dusit Niyato, Jiawen Kang, Zehui Xiong, Dong In Kim, Xuemin Sherman Shen, and Khaled B Letaief. 2025. Enhancing physical layer communication security through generative ai with mixture of experts. *IEEE Wireless Communications*.

A A Proof of Output Variation Approximation

This section provides the proof for the approximation of the output variation resulting from retaining the expert set S_l in the l-th MoE layer, as described in Equation (3).

The pruned routing weights are re-normalized based on the original weights. Formally, this can be expressed as:

$$[G'_l(\boldsymbol{H}_l)]_i = \frac{[G_l(\boldsymbol{H}_l)]_i}{\sum_{j\in S_l}^{|S_l|}[G_l(\boldsymbol{H}_l)]_j}, \quad \forall i \in S_l \tag{6}$$

We now expand the second term in Equation 3:

$$\begin{aligned}&\sum_{k\in S_l}^{|S_l|}\left([G_l(\boldsymbol{H}_l)]_k - [G'_l(\boldsymbol{H}_l)]_k\right)\odot E_{l,k}(\boldsymbol{H}_l)\\ &= \sum_{k\in S_l}^{|S_l|}[G_l(\boldsymbol{H}_l)]_k\left(1-\frac{1}{\sum_{j\in S_l}^{|S_l|}[G_l(\boldsymbol{H}_l)]_j}\right)\odot E_{l,k}(\boldsymbol{H}_l)\end{aligned} \tag{7}$$

Since our pruning strategy effectively identifies critical experts and allocates a relatively uniform pruning threshold across layers, we approximate $\sum_{j\in S_l}^{|S_l|}[G_l(\boldsymbol{H}_l)]_j \approx 1$. This simplifies the term to:

$$\left(1-\frac{1}{\sum_{j\in S_l}^{|S_l|}[G_l(\boldsymbol{H}_l)]_j}\right)\approx 0 \tag{8}$$

Thus, we can conclude that $[G_l(\boldsymbol{H}_l)]_k\left(1-\frac{1}{\sum_{j\in S_l}^{|S_l|}[G_l(\boldsymbol{H}_l)]_j}\right) \approx 0$, implying that $[G_l(\boldsymbol{H}_l)]_k \approx [G'_l(\boldsymbol{H}_l)]_k$.

B Details of Benchmark Evaluations

We conduct extensive comparative experiments to evaluate our proposed dynamic pruning approach against baseline models. The evaluation is performed on multiple popular benchmarks using the OpenCompass LLM evaluation framework. The benchmarks used for evaluation include:

- **ARC-e** is a dataset consisting of 3,000 elementary-level science questions in a multiple-choice format. We report the zero-shot accuracy on ARC-e and use the prompt *"Question: {question} A: {textA} B: {textB} C: {textC} D: {textD}. Answer:"*.

- **ARC-c** includes challenging science questions compared to ARC-e, designed to assess commonsense reasoning and factual knowledge. We report zero-shot accuracy on ARC-c and use the same prompt as ARC-e.

- **BoolQ** is a dataset 15,942 contains yes/no questions based on passages from various sources. We report zero-shot accuracy on BoolQ and use *"Passage: {passage} Question: {question}? A. Yes B. No. Answer:"* as the prompt.

- **HellaSwag** is a challenging benchmark designed for evaluating commonsense reasoning. Given an incomplete context, the model must predict the most plausible ending. We report accuracy on HellaSwag and use *"{ctx} Question: Which ending makes the most sense? A. {A} B. {B} C. {C} D. {D} You may choose from 'A', 'B', 'C', 'D'. Answer:"* as the prompt.

- **MMLU** is a benchmark designed to evaluate the general knowledge of models, consisting of 57 tasks across various domains, including mathematics, professional law, and sociology. We report the accuracy on multiple tasks within MMLU and employ task-specific prompts for each evaluation.

- **WinoGrande** is a dataset designed to assess a model's ability to resolve coreference in complex sentences, consisting of 44,000 problems. We report accuracy on WinoGrande and use *"Which of the following is a good sentence: A. {opt1} B. {opt2} Answer:"* as the prompt.

C More Experimental Details

C.1 More Experimental Results

This section presents additional comparative results between our method and the baselines under the configuration where expert parameters are compressed by 33.3%. Tables 6 and 7 extend Tables 1 and 2, respectively.

From Table 6, we observe similar trends to those in Section 5.2: when the expert parameter is reduced by 33%, our method achieves the best performance across most benchmarks. However, due to significant distributional differences between the upstream calibration dataset and the benchmarks, methods such as BSP, MC MoE, and Expert Sparsity exhibit considerable performance discrepancies on some datasets compared to our approach. For example, using Mixtral 8×7B Instruct as the backbone, MC MoE performs only 1.62% worse than our method on BoolQ, while

Model	Params↓	Method	ARC-e	ARC-c	BoolQ	HellaSwag	MMLU	Winogrande	Average
Mixtral Instruct	33.3%	BSP	70.32	46.70	54.46	36.96	34.92	48.93	48.72
		EEP	83.53	70.38	78.32	61.83	47.46	**59.62**	67.19
		MC MoE	78.27	65.49	76.70	53.80	47.03	56.99	63.05
		Expert Sparsity	83.09	68.15	**79.79**	62.99	48.31	58.49	66.80
		PREP	**88.54**	**77.25**	78.32	**68.93**	**56.52**	57.54	**71.18**
Mixtral	33.3%	BSP	63.68	45.49	56.88	25.19	29.28	34.27	42.47
		EEP	74.81	61.83	70.32	37.13	45.16	52.72	57.00
		MC MoE	72.11	61.04	64.40	38.23	47.31	51.78	55.81
		Expert Sparsity	72.67	58.11	63.98	34.71	48.82	51.38	54.95
		PREP	**83.25**	**70.38**	**71.82**	**44.49**	**60.22**	**56.21**	**64.40**

Table 6: Zero-shot performance comparison of different methods under varying expert parameter compression ratios. "Params↓" represents the reduction radio in expert parameters. "Average" is calculated among six benchmarks. The best results are shown in **bold**.

Model	Params↓	Method	Wiki	C4
Mixtral Instruct	33.3%	BSP	9.71	19.45
		EEP	6.04	10.25
		MC MoE	5.54	**9.05**
		Expert Sparsity	5.46	8.98
		PREP	**4.91**	9.49
Mixtral	33.3%	BSP	8.21	21.85
		EEP	7.14	13.39
		MC MoE	5.65	11.92
		Expert Sparsity	5.53	10.88
		PREP	**5.47**	**10.22**

Table 7: Perplexity evaluation on WikiText2 and C4. The best results are shown in **bold**.

it exhibits a 15.13% performance gap on the HellaSwag benchmark.

Table 7 demonstrates that our method consistently achieves the lowest perplexity, confirming its retention of the original language modeling capabilities. Thanks to the pruning strategy based on the C4 training set, both MC MoE and Expert Sparsity methods exhibit similar performance to ours.

C.2 Implementation Details for Expert Predictor

In this section, we illustrate the implementation details of the expert predictor, which is a comparative variant for expert evaluation strategy in Section 5.3.1.

Training Data Collection As detailed in Section 5.3.1, the goal of the expert predictor is to effectively predict the relative importance of each expert for each sample, such that the difference between the modified output and the original distribution is minimized. However, the computational cost of enumerating all possible expert combinations is prohibitively high. To address this, we first assess the importance of each expert for the current sample using the method proposed in (Yang et al., 2024). Subsequently, we rank the experts based on their scores and use their rank and scores as labels. Intuitively, we choose the attention weights, hidden states, routing weights, and layer indices from each layer as the input to the predictor.

Details of Expert Predictor (a) Network Architecture: (i) Layer Idx embedding: An embedding layer with 32 embedding units (corresponding to the number of layers) and an embedding dimension of 128 for the layer index; (ii) Attention Gate layer: A linear layer with an input size of 1 and a hidden size of 128 for attention weights; (iii) Routing Weight Gate layer: A linear layer with an input size of 8 (representing the number of local experts) and a hidden size of 128 for routing weights; (iv) Input Layer: a linear with an input size of 4096 and a hidden size of 128 for hidden state; (v) CLS Embedding: A randomly initialized parameter with a hidden dimension of 128 for the CLS token; (vi) Encoder: A Transformer (Vaswani, 2017) comprising two layers, each with 2 attention heads and a hidden size of 128; (vii) Output layer: A linear layer with an input size of 4096 and a hidden size of 8 (representing the number of local experts) for the CLS token. (b) Training Process: Due to the limited size of the collected dataset, we augment the data using a pairwise approach (Raziperchikolaei and Chung, 2024) to maximize its utility while reducing the risk of overfitting. Specifically, pairs of samples are formed within each batch, and the model is trained using hinge loss as the objective function to optimize the overall predictions; (iii) Hyperparameters: We use the Adam optimizer

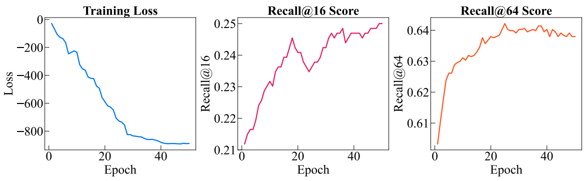

Figure 5: Training loss of the expert predictor and recall score on the validation set during training.

Params↓	Expansion Point	MMLU	HellaSwag
25.0%	Constant	56.72	67.33
	Gaussian Noise	46.71	50.06
	Cluster Center	57.24	67.38
	Mean State (Ours)	**57.92**	**71.76**
33.3%	Constant	54.64	64.74
	Gaussian Noise	42.35	24.67
	Cluster Center	56.13	66.39
	Mean State (Ours)	**56.52**	**68.93**
40.0%	Constant	52.02	61.46
	Gaussian Noise	40.73%	24.96
	Cluster Center	54.51	**61.71**
	Mean State (Ours)	**54.74**	67.17

Table 8: Evaluation results of different Taylor expansion points for linearized expert importance. The best results are shown in **bold**.

with a learning rate of 0.001, trained for 50 epochs, and apply a decay rate of 0.95 every 1000 steps. The collected dataset is split into a training set and a validation set in a 7:3 ratio. (c) Training Results: Figure 5 reports the loss on the training set, as well as the recall@16 and recall@64 on the validation set during the training process of the expert predictor. It can be observed that the recall@16 and recall@64 on the validation set converge to approximately 25% and 64%, respectively, which is below the desired performance. This suggests that capturing the relative importance of experts based on input using a simple lightweight trainable module is challenging.

C.3 More Experiment Settings

In this section, we provide additional experimental settings to facilitate the reproduction of our results.

Hardware Setup The hardware platform used in this experiment consists of 8 NVIDIA RTX 3090 GPUs and 1 NVIDIA L20 GPU. The CPU is an AMD Ryzen Threadripper 3960X 24-core Processor, featuring 48 logical CPU cores, with each core supporting 2 threads.

Model Quantization Settings We apply Half-Quadratic Quantization (Badri and Shaji, 2023) to quantize the backbone model. Specifically, for Mixtral 8×7B and Mixtral 8×7B Instruct, we quantize the transformer attention block parameters to 4 bits with a group size of 64 and the experts in the MoE layer to 3 bits with a group size of 64. For DeepSeek-V2-Lite-Chat, we quantize both the transformer attention block and the MoE block, including shared and independent experts, to 4 bits, with a group size of 64.

Params↓	Faiss Index	MMLU	HellaSwag
25.0%	IndexFlatIP	58.37	71.84
	IndexIVFFlat (Ours)	57.92	71.76
33.3%	IndexFlatIP	56.55	69.07
	IndexIVFFlat (Ours)	56.52	68.93
40.0%	IndexFlatIP	55.60	67.67
	IndexIVFFlat (Ours)	54.74	67.17

Table 9: Ablation results on different Index methods.

D Influence of Taylor Expansion Point

In this subsection, we investigate how the choice of the Taylor expansion point for the linearized expert importance affects overall performance. Specifically, we explore several variants for selecting different Taylor expansion points: **1)** Constant: The expansion point is a constant vector of all ones; **2)** Gaussian Noise: The expansion point is a random vector drawn from a Gaussian distribution; **3)** Cluster Center: The expansion point is the cluster center (k=3) of the hidden states collected from each layer of the RedPajama dataset, obtained using the K-means algorithm (Tarpey, 2007).

Table 8 shows how different Taylor expansion points affect model performance across various expert parameter compression ratios on the MMLU and HellaSwag benchmarks. The results reveal that **the mean of the hidden state as the expansion point yields superior overall performance.** Notably, using the cluster center as the expansion point leads to suboptimal results. We attribute this to the fact that the cluster center tends to overfit the calibration set distribution, leading to poor generalization in downstream benchmarks.

E Effects of Faiss Index

To boost search efficiency, we replace the basic *IndexFlatIP* method, which does exact searches when building the index, with the faster but approximate *IndexIVFFlat* method (Douze et al., 2024). We show how both methods perform in Table 9 and also check how quickly they work.

The results demonstrate that the use of the accelerated index does not significantly impact overall model performance compared to the exact index, confirming its viability as an alternative in our approach. Furthermore, constructing a more efficient Faiss index reduces the search time to approximately 1.3×10^{-3} seconds, compared to 1.04×10^{-2} seconds for the exact search, resulting in an $8.0\times$ speedup.