1 Additional Method Details

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 (Section 1.1). 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 (Section 1.2). Finally, we introduce a sample reordering strategy to extend our method to batch inference scenarios (Section 1.3).

1.1 Efficient Important Expert Evaluation

1.1.1 Definition of Expert Importance

We define the importance of the k-th expert in the l-th MoE layer as the maximum activation value of its output across all n tokens:

$$\sigma_{l,k} = \max_{i\in[n],j\in[d]} [E_{l,k}(\boldsymbol{H}_l)]_{i,j}. \tag{1}$$

This metric bounds potential output variation when pruning experts; higher values of $\sigma_{l,k}$ indicate experts whose removal could cause significant output changes and should therefore be retained.

To theoretically justify this intuition, we analyze the output variation from expert pruning. Specifically, the output variation $\Delta\boldsymbol{Y}_l$ is decomposed into:

$$\begin{aligned}\Delta\boldsymbol{Y}_l = \boldsymbol{Y}_l - \hat{\boldsymbol{Y}}_l &= \sum_{k\notin S_l}^{K-|S_l|} [G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l) \\ &+ \sum_{k\in S_l}^{|S_l|} ([G_l(\boldsymbol{H}_l)]_k - [G'_l(\boldsymbol{H}_l)]_k) \odot E_{l,k}(\boldsymbol{H}_l) \\ &\approx \sum_{k\notin S_l}^{K-|S_l|} [G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l).\end{aligned} \tag{2}$$

where the pruned routing weights $G'_l(\boldsymbol{H}_l)$ are re-normalized based on the original weights. Formally, this can be expressed as:

$$[G'_l(\boldsymbol{H}_l)]_i = \frac{[G_l(\boldsymbol{H}_l)]_i}{\sum_{j\in S_l}^{|S_l|} [G_l(\boldsymbol{H}_l)]_j}, \quad \forall i \in S_l \tag{3}$$

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

$$\Delta\boldsymbol{Y}_l \approx \sum_{k\notin S_l}^{K-|S_l|} [G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l). \tag{4}$$

where the impact of expert pruning on the model's output is estimated by the weighted sum of the outputs of the pruned experts. Following the derivation, we can upper bound the output of the pruned experts (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:

$$||[G_l(\boldsymbol{H}_l)]_k \odot E_{l,k}(\boldsymbol{H}_l)||_2 \le ||[G_l(\boldsymbol{H}_l)]_k \cdot \sigma_{l,k} \cdot d||_2, \tag{5}$$

where d represents the feature dimension of the hidden state. Therefore, 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.

1.1.2 Fast Search of Important Experts

To efficiently evaluate expert importance, we propose a fast search method that significantly reduces computation cost. Specifically, we first introduce an alternative of expert importance by applying max pooling over the hidden states. Next, we linearize the alternative using a first-order Taylor expansion. Finally, we precompute the Jacobian matrix for each expert and construct a compact search index, which supports fast retrieval of the most important experts on the CPU during inference.

The maximum activation value-based expert importance additionally confers an advantage: it enables efficient computation without incurring GPU memory overhead. Specifically, the proposed expert importance introduces an alternative that enables subsequent efficient CPU-based search through an input compression operation as:

$$\tilde{\sigma}_{l,k} = \max_{i\in[n]} [E_{l,k}\left(\boldsymbol{h}_{l,\max}\right)]_i, \tag{6}$$

where $\boldsymbol{h}_{l,\max} = \text{max-pool}(\boldsymbol{H}_l)$ is the max-pooling over $\boldsymbol{H}_l$ on feature dimension. Specifically, we first apply max-pooling to the input and then compute the final maximum value over the resulting vector. This approach retains critical input features while enabling fast evaluation of expert importance based on a compressed input representation.

To enable efficient evaluation of expert importance without heavy 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:

$$\tilde{\sigma}_{l,k} \approx \max_j \left[E_{l,k}(\boldsymbol{x}_0) + \left(\frac{\partial E_{l,k}(\boldsymbol{x})}{\partial \boldsymbol{x}}|_{\boldsymbol{x}=\boldsymbol{x}_0}\right)^\top (\boldsymbol{h}_{l,\max} - \boldsymbol{x}_0)\right]_j, \tag{7}$$

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 general-purpose text corpus, using this value as the Taylor expansion point for each layer. We evaluate the different expansion points in section 2.4.2.

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 [1]. 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.

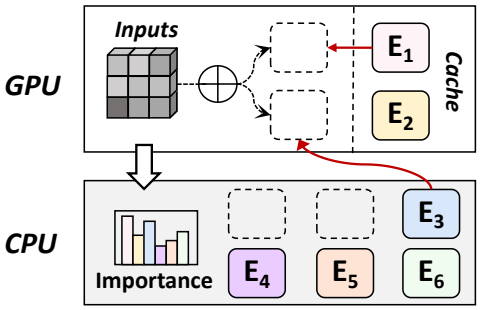

Fig. 1 Illustration of Layer-wise Loading of Experts. For each layer, the highest-importance experts are loaded from CPU or cache to perform the forward pass.

1.1.3 Discussion

A straightforward way to measure expert importance is to sum each expert's router scores across all tokens. While this method appears intuitive, it performs poorly in practice. The key limitation lies in the fact that different tokens often favor different experts, leading to highly diverse routing preferences. **When these scores are aggregated, the variation across tokens is smoothed out, making all experts appear similarly important.** As a consequence, most experts appear similarly important, which weakens the signal required for pruning. As shown in our empirical analysis (Section 2.4.1), our proposed metric clearly outperforms such routing weight-based pruning methods.

1.2 Adaptive Layer-wise Expert Loading

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

1.2.1 Across-Layer Importance Analysis

Prior work has established that distinct layers in LLMs serve specialized functional roles [2, 3]. 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 [4], we propose an intuitive metric I_l for the l-th layer importance: the Jensen-Shannon divergence [5] between the model's original output distribution p_{ori} and its modified distribution p_l when excluding the l-th layer as:

$$I_l = \mathbb{E}_{\boldsymbol{X}\sim\mathcal{D}}\left[\frac{1}{2}D_{\text{KL}}(p_{\text{ori}}(\boldsymbol{X})||p_m(\boldsymbol{X})) + \frac{1}{2}D_{KL}(p_l(\boldsymbol{X})||p_m(\boldsymbol{X}))\right], \tag{8}$$

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 2.4.4.

1.2.2 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** [6, 7]. 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 and loads them into the GPU memory, while offloading other experts to main memory. Leveraging the similarity of adjacent tokens' hidden states, we employ an expert cache per layer with a Least Recently Used (LRU) policy [8] to retain recently active experts, thereby minimizing delays from frequent memory swaps and enhancing inference efficiency.

1.3 Batch Inference

The proposed input-aware pruning method is designed to dynamically tailor expert selection for each individual input. However, this flexibility poses a challenge in the context of batch inference, where hardware efficiency typically requires all inputs within a batch to share the same set of active experts. The constraint can significantly diminish the effectiveness of input-aware pruning, particularly when the inputs within a batch are heterogeneous [9]. To address this, we introduce a sample reordering strategy that enables effective batching without sacrificing the advantages of input-aware expert selection. The core idea is to group together inputs with similar hidden state representations, as they are more likely to activate similar experts. We achieve this by formulating the sample ordering as a Traveling Salesman Problem (TSP) [10]. In this formulation, each input is a node, and the distance between nodes is defined by the dissimilarity of their embeddings. The objec-

Algorithm 1: Input-Aware Sample Reordering for Efficient Batching

Input:
- Stream of input samples $\{X_1, X_2, \ldots\}$
- Batch size B
- Distance threshold δ

Output: Reordered sub-batches $\mathcal{G}_1, \mathcal{G}_2, \ldots$ each with $\leq B$ samples

Step 1: Collect and Embed Samples
Collect a temporary batch $\mathcal{B} = \{X_1, \ldots, X_B\}$
Compute embeddings $\mathcal{H} = \{h_i = \text{Embed}(X_i)\}_{i=1}^{B}$
Compute pairwise distances $\mathbf{D}_{ij} = \|h_i - h_j\|_2$

Step 2: Greedy TSP Ordering
Initialize route $\mathcal{R} = [\arg\min_i \sum_j \mathbf{D}_{ij}]$ // start from medoid
while $|\mathcal{R}| < B$ **do**
  Append $\arg\min_{j\notin\mathcal{R}} \mathbf{D}_{\mathcal{R}[-1],j}$ to $\mathcal{R}$
end

Step 3: Distance-Constrained Grouping
Initialize $\mathcal{G} = \{\}$; current = $[\mathcal{R}[1]]$
for $i = 2$ ***to*** B **do**
  Compute mean distance $\bar{d} = \frac{1}{|\text{current}|}\sum_{k\in\text{current}} \mathbf{D}_{\mathcal{R}[i],k}$
  if $\bar{d} \leq \delta$ ***and*** $|current| < B$ **then**
    Append $\mathcal{R}[i]$ to current
  end
  else
    Append current to $\mathcal{G}$; current = $[\mathcal{R}[i]]$
  end
end
Append final current to $\mathcal{G}$
return sub-batches $\mathcal{G}_1, \mathcal{G}_2, \ldots$

Table 1 Statistics of the experimental datasets, including the number of samples in evaluation, the average/maximum/minimum token counts per sample, and their corresponding task fields.

Dataset	Sample	Avg Tokens	Max Tokens	Task Field
ARC-easy	2365	59	198	Grade-school Science
ARC-challenge	1165	68	191	Grade-school Science
BoolQ	3270	147	1063	Commonsense
HellaSwag	10042	205	351	Commonsense
MMLU	14327	107	994	Multiple Fields (Math, Law)
WinoGrande	1267	62	93	Commonsense
WikiText2	36718	64	841	Wikipedia Corpus
C4	356317	478	57966	Common Crawl Corpus

tive is to find the shortest path that traverses all nodes, arranging the inputs in an order of maximum consecutive similarity.

Given the NP-hard complexity of TSP, we adopt an efficient greedy algorithm [11] to approximate the optimal solution. The algorithm initializes the sequence with the input sample exhibiting the lowest total pairwise similarity to other inputs. Subsequently, it iteratively extends the sequence by appending the unvisited sample that is most similar to the current final sample in the sequence. This process continues until all samples are included in the sequence, thereby maximizing the similarity between successive samples. After constructing the optimized sequence, we partition it into fixed-size batches based on a predefined batch size.

2 Additional Experiments Details

To evaluate the effectiveness of PREP, we conduct extensive experiments on benchmark datasets derived from real-world scenarios. Our investigation primarily focuses on assessing the performance and inference efficiency of PREP, along with a detailed analysis of the strategies and components integrated into our method.

2.1 Datasets and Evaluation Metrics

- **Datasets**

To comprehensively compare the performance of different methods and validate our findings, we conduct extensive experiments on several general benchmarks to assess models' fundamental capabilities. The general benchmarks include ARC-easy and ARC-challenge (a grade-school-level multiple-choice science dataset partitioned into Easy and Challenge Sets, covering basic comprehension and complex inferential questions respectively) [12], BoolQ (a commonsense Boolean QA dataset with questions paired with the provided real-world text paragraphs) [13], HellaSwag (a commonsense inference benchmark centered on sentence completion tasks with contextually consistent daily activity endings) [14], MMLU (a university-level multi-disciplinary benchmark covering 57 fields to evaluate zero-shot reasoning and cross-domain transfer) [15], and WinoGrande (a large-scale pronoun resolution dataset designed to test robust commonsense reasoning via bias-reduced ambiguous contexts) [16]. The evaluation follows the settings of OpenCompass [17], a widely used LLM evaluation framework. Furthermore, we supplement our evaluation with perplexity analysis to assess the model's language modeling capability on the WikiText2 (a Wikipedia-derived corpus for pre-training and language modeling tasks) [18] and C4 (a large-scale English corpus from Common Crawl, filtered for quality and covering diverse web domains) [19] public datasets. Statistics of the experimental datasets are shown in Table 1.

- **Metrics**

The evaluation metrics are used to quantify performance and efficiency:

- **Accuracy**: Defined as the proportion of correctly predicted outputs among all test samples, used to evaluate performance on downstream tasks (ARC-e, ARC-c, BoolQ, HellaSwag, MMLU, WinoGrande).
- **Perplexity (PPL)**: Measures language modeling quality, with lower values indicating stronger ability to predict the next token in a sequence. The formula is given by:

$$\mathrm{PPL} = \exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log p(x_i|x_{<i})\right) \tag{9}$$

  where N is the total number of tokens in the sequence, and $p(x_i|x_{<i})$ is the model's predicted probability of the i-th token given all preceding tokens $x_{<i}$.
- **Latency**: Quantifies inference efficiency, defined as the time required to generate one token. It reflects the model's real-time response capability during inference.

2.2 Baseline Methods and Settings

- **Baselines**

We compare our method with the following expert compression baselines:

- **BSP** [20] calculates each expert's importance score via a predictor to allocate differentiated weight bit-widths. The predictor predicts cosine similarity between an MoE block's input and output, and uses calibration data to compute each block's average score for bit allocation.
- **Expert Sparsity** [21] proposes a post-training approach for expert pruning: at the layer level, it enumerates all possible expert combinations and retains the optimal one by minimizing Mean Squared Error (MSE) loss between the outputs of the original MoE model and the pruned model.
- **EEP** [22] is a gradient-free evolutionary pruning strategy for MoE models. It first designs a parameter space for router mapping and expert merging. Subsequently, it searches within these weight components to preserve the most prominent experts, extracts knowledge from the pruned experts, and consolidates this knowledge into the retained ones.
- **MC MoE** [23] first formulates adaptive bit-width allocation as a Linear Programming problem via pre-loading mixed-precision quantization, with the objective function balancing multi-factors (the activation frequency and the sum of routing weights) that reflect expert importance.

Table 2 Performance comparison of different methods under varying expert parameter compression ratios. "Params↓" represents the reduction ratio in expert parameters. "Wino." is the short format of "Winogrande". "Average" is the average performance averaged across six benchmarks. The base model serves as the upper limit. The best results are highlighted in **bold**.

Model	Params↓	Method	ARC-e	ARC-c	BoolQ	HellaSwag	MMLU	Wino.	Average%
Mixtral Instruct	-	Base	91.12	81.20	83.98	71.22	63.91	62.45	75.65
	25.0%	BSP	83.09	50.52	59.72	42.72	46.46	52.09	55.77 19.8↓
		EEP	86.94	73.67	83.43	63.99	49.25	60.24	69.59 6.1↓
		MC MoE	84.74	72.19	82.39	56.38	55.01	58.56	68.21 7.4↓
		Expert Sparsity	84.21	72.36	**85.35**	63.83	51.14	59.83	69.45 6.2↓
		PREP	**90.19**	**78.63**	80.85	**71.76**	**57.92**	**61.56**	**73.48** 2.2↓
	33.3%	BSP	70.32	46.70	54.46	36.96	34.92	48.93	48.72 26.9↓
		EEP	83.53	70.38	78.32	61.83	47.46	**59.62**	67.19 8.5↓
		MC MoE	78.27	65.49	76.70	53.80	47.03	56.99	63.05 12.6↓
		Expert Sparsity	83.09	68.15	**79.79**	62.99	48.31	58.49	66.80 8.9↓
		PREP	**88.54**	**77.25**	78.32	**68.93**	**56.52**	57.54	**71.18** 4.5↓
	40.0%	EEP	81.36	68.62	72.04	58.23	43.15	55.36	63.13 12.5↓
		MC MoE	75.73	64.38	73.21	44.57	44.77	53.67	59.39 16.3↓
		Expert Sparsity	79.32	64.29	**79.94**	59.33	45.13	**58.09**	64.35 11.3↓
		PREP	**87.99**	**73.73**	77.55	**67.17**	**54.74**	57.70	**69.81** 5.8↓
Mixtral	-	Base	89.26	72.56	78.45	50.44	66.37	57.81	69.15
	25.0%	BSP	76.87	55.49	59.39	34.77	36.32	48.30	51.86 17.3↓
		EEP	79.53	64.27	71.84	42.62	50.41	54.23	60.48 8.7↓
		MC MoE	78.22	64.72	62.32	42.60	53.80	54.85	59.42 9.7↓
		Expert Sparsity	76.79	61.03	68.62	41.68	53.36	54.22	58.28 10.9↓
		PREP	**85.71**	**70.82**	**73.73**	**46.67**	**64.09**	**55.88**	**66.15** 3.0↓
	33.3%	BSP	63.68	45.49	56.88	25.19	29.28	34.27	42.47 26.7↓
		EEP	74.81	61.83	70.32	37.13	45.16	52.72	57.00 12.2↓
		MC MoE	72.11	61.04	64.40	38.23	47.31	51.78	55.81 13.3↓
		Expert Sparsity	72.67	58.11	63.98	34.71	48.82	51.38	54.95 14.2↓
		PREP	**83.25**	**70.38**	**71.82**	**44.49**	**60.22**	**56.21**	**64.40** 4.8↓
	40.0%	EEP	70.36	60.64	65.04	34.62	43.52	51.75	54.32 14.8↓
		MC MoE	64.71	56.64	60.06	26.31	41.74	50.52	50.00 19.2↓
		Expert Sparsity	68.67	55.19	62.35	31.63	46.67	50.75	52.54 16.6↓
		PREP	**82.66**	**69.27**	**69.66**	**42.11**	**58.23**	**54.22**	**62.69** 6.5↓

• **Settings**

For the primary evaluation, we adopt Mixtral 8 × 7B and Mixtral 8 × 7B Instruct [24] as our backbone models. DeepSeek-V2-Lite-Chat [25] is introduced as an additional backbone to further validate the effectiveness of our method, with detailed analyses provided in Section 2.5. Both Mixtral 8 × 7B and Mixtral 8 × 7B Instruct comprise 32 transformer layers, where each layer integrates an MoE block with 8 experts and employs a top-2 expert routing strategy. In contrast to the Mixtral models (which follow a uniform MoE design across all layers), DeepSeek-V2-Lite-Chat consists of 27 transformer layers: the first layer adopts a dense feed-forward network (FNN), while the subsequent 26 layers incorporate MoE blocks. Each MoE block in DeepSeek-V2-Lite-Chat includes 2 shared experts and 64 independent experts, utilizing a top-6 routing strategy. For Perplexity calculation, we follow [26] and [27] to ensure result reliability: First, 128 samples are randomly sampled from the test set, and all samples are concatenated into a single continuous text. Subsequently, a sliding window strategy is employed to split the concatenated text into segments, with a window length of 2048 tokens and a sliding stride of 512 tokens, ensuring the contextual integrity of the text within each window. Finally, the model performs autoregressive prediction for each token in each window, and the final PPL value is computed as the exponential of the average negative log-likelihood. For Hardware Setup, the experiments are conducted on NVIDIA 3090 GPU, one of the most widely used high-end consumer graphics cards. The CPU is an AMD Ryzen Threadripper 3960X 24-core Processor, featuring 48 logical CPU cores, with each core supporting 2 threads.

2.3 Main Experimental Results

We present model performance on standard benchmarks (Table 2), their perplexity on language modeling (Figure 2), and inference latency across varying input lengths (Figure 3). From the results, we have the following observations:

1) PREP achieves the best performance, outperforming all baselines across all benchmarks. Table 2 shows that PREP achieves 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. Notably, this phenomenon is particularly pronounced on MMLU—a benchmark encompassing 57 diverse domains including mathematics, chemistry, and computer science. For illustration, on the MMLU benchmark using the Mixtral Instruct model at 33.3% parameter reduction, PREP outperforms MC MoE and Expert Sparsity by 9.49% and 8.21%, respectively. In contrast, EEP leverages task-specific pruning by directly optimizing configurations

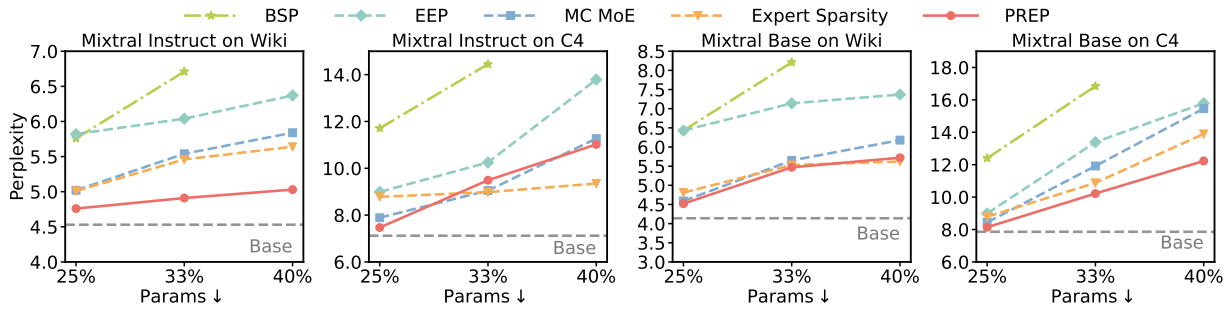

Fig. 2 Perplexity results of different methods on WikiText2 and C4 datasets with varying expert parameter compression ratios based on Mixtral 8 × 7B Instruct and Mixtral 8 × 7B. "Base" denotes the performance of the parameter-uncompressed base model, which is indicated by the gray dashed line in the figure and corresponds to the target perplexity.

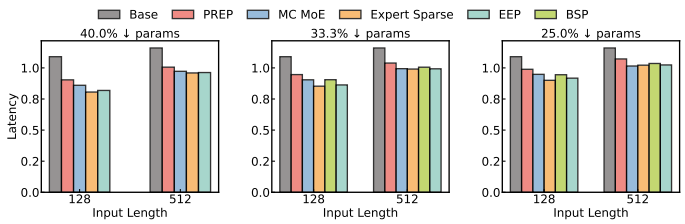

Fig. 3 Inference speed comparison for different input lengths under various parameter pruning ratios. "Base" denotes the performance of the uncompressed baseline model; note that the pruning ratio does not apply to Base (as it is unpruned), and Base's latency is only affected by input length.

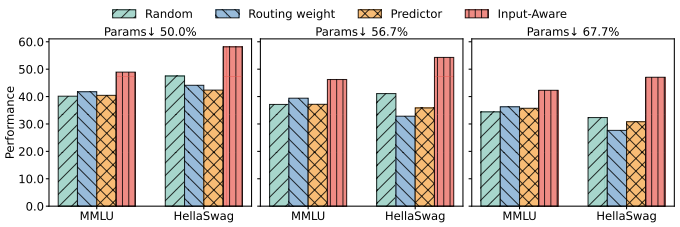

Fig. 4 Performance comparison of different expert evaluation strategies for dynamic pruning.

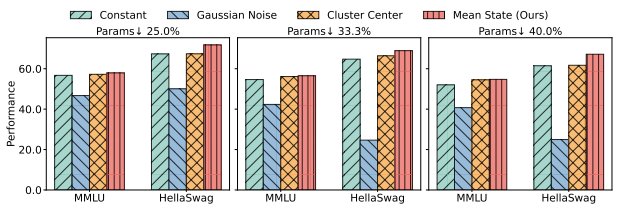

Fig. 5 Evaluation results of different Taylor expansion points for linearized expert importance.

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. The perplexity results for various methods on WikiText2 and C4 are shown in Figure 2. It can be observed that under different corpus modeling scenarios and compressed parameter settings, PREP exhibits performance closest to the Base model compared to other MoE compression methods, demonstrating its robustness across diverse conditions. Notably, on the C4 dataset— which shares distributional similarities with the pruning-referenced calibration data used in MC MoE and Expert Sparsity—our method still outperforms these methods. This highlights that the proposed method maintains considerable performance even under in-domain conditions. In contrast, EEP significantly underperforms, as its pruning results are derived from narrow data subsets.

3) PREP exhibits the latency optimization when modeling inputs of varying lengths. As shown in Figure 3, our method delivers a 1.2 × inference speedup over the base model. This efficiency gain stems from two key innovations: 1) a carefully designed expert loading strategy that minimizes redundant overhead, and 2) an effective important expert evaluation strategy that reduces the computation of redundant experts. 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.

2.4 Analysis Experiments

2.4.1 Validity of Expert Evaluation Strategy

This experiment evaluates the effectiveness of our proposed layer-wise expert evaluation strategy. In particular, we modify the proposed expert evaluation strategy with the following variants:

- Random Scoring: Experts are randomly scored for retention.
- Routing Weight-based evaluation: Experts are evaluated based on their cumulative routing weight contributions.
- Predictor-based evaluation: Experts are evaluated by a trained predictor. The predictor, consisting of a two-layer Transformer encoder and two MLP layers, is trained on labels derived from expert importance rankings with pairwise data augmentation, using hinge loss as the objective function.

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. We summarize the results in Figure 4. 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**

Table 3 Performance comparison between Base and our method under varying expert parameter compression ratios, with Base serving as the upper limit. "Wino." is the short format of "Winogrande". "Average" is the average performance averaged across six benchmarks.

Model	Method	Params↓	ARC-e	ARC-c	BoolQ	HellaS.	MMLU	Wino.	Average%
DeepSeek	Base	0	80.88	68.24	73.57	65.15	50.20	57.77	65.97
	PREP	25.0%	78.04	62.63	71.84	62.45	47.42	54.61	62.83
		33.3%	74.84	61.55	70.02	62.96	46.30	51.35	61.17
		40.0%	72.13	59.00	68.02	51.59	41.17	52.30	57.36

Table 4 Ablation results on different Faiss Index methods. "Search Time" denotes the total Faiss search time per layer for computing the importance of each expert.

Params↓	Faiss Index	MMLU	HellaSwag	Search Time
25.0%	IndexFlatIP	58.37	71.84	1.09×10^{-2}
	IndexIVFFlat (Ours)	57.92	71.76	1.36×10^{-3}
33.3%	IndexFlatIP	56.55	69.07	1.05×10^{-2}
	IndexIVFFlat (Ours)	56.52	68.93	1.31×10^{-3}
40.0%	IndexFlatIP	55.60	67.67	1.01×10^{-2}
	IndexIVFFlat (Ours)	54.74	67.17	1.27×10^{-3}

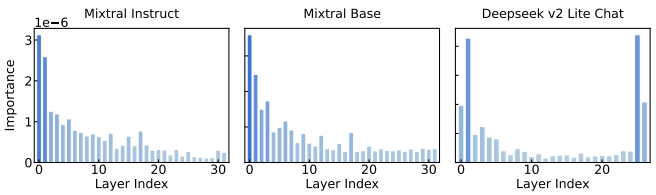

Fig. 6 Layer importance for Mixtral-8×7B Instruct, Mixtral-8×7B Base and DeepSeek-V2-Lite-Chat. The y-axis represents the importance score of each layer with respect to the final output, which quantifies the Jensen-Shannon divergence between the final output of the original model and that of the model with the corresponding layer pruned.

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**.

Model	Method	Params↓	Latency_{128}	Latency_{512}
DeepSeek	Base	-	0.71	0.82
	PREP	25.0%	0.54	0.63
		33.3%	0.51	0.61
		40.0%	**0.49**	**0.58**

methods and our strategy increase as parameter compression intensifies, exposing fundamental limitations in these variants, namely, their inability to dynamically identify input-critical experts and inherent robustness deficiencies.

2.4.2 Influence of Taylor Expansion Point

In this subsection, we investigate how the choice of Taylor expansion point for the linearized expert importance affects overall performance. Specifically, we explore several variants for selecting different Taylor expansion points:

- Constant: The expansion point is a constant vector of all ones.
- Gaussian Noise: The expansion point is a random vector drawn from a Gaussian distribution.
- 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 [28].

Figure 5 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.

2.4.3 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 [29]. We present the performance and search efficiency of both methods in Table 4.

The results demonstrate that the accelerated index does not significantly degrade overall model performance compared to the exact index, owing to the robust optimization capabilities of the Faiss library and the discriminative representations of the extracted expert parameters. Furthermore, constructing a more efficient Faiss index reduces the search time to approximately 1.30×10^{-3} seconds, compared to 1.04×10^{-2} seconds for the exact search, resulting in an 8.0× speedup.

2.4.4 Analysis of Across-Layer Importance

Following the definition of layer importance in Section 1.2, we analyze the importance for Mixtral 8×7B Instruct, Mixtral 8×7B Base and DeepSeek-V2-Lite-Chat on the RedPajama dataset, as shown in Figure 6. It can be observed that these **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 series models exhibit 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 models, we partition their 32 MoE layers into four groups. The number of retained experts across these groups follows a 2:2:1:1 ratio, with retention thresholds uniformly assigned across the layers within each group. For the DeepSeek model, we partition its first 24 MoE layers into four groups, and the remaining 2 MoE layers form an independent fifth group. The number of retained experts across the five groups adheres to a 2:2:1:1:1 ratio, with retention thresholds evenly distributed among the layers of each group.

2.5 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**

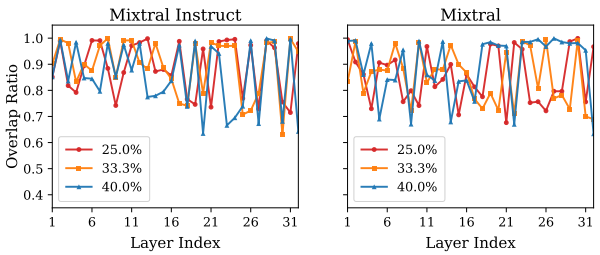

Fig. 7 Overlap ratio between Top-m expert sets selected by the exact max-activation score and its Taylor-linearized Approximation.

corresponding code. For a baseline comparison, we use a uniform 4-bit quantization strategy for each expert and fully load the expert 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 5 (latency vs. input length) and Table 3 (accuracy rate performance). From the results, we have the following observations: **1) The proposal effectively improves inference speed over the base method.** In Table 5, pruning 25.0% 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. **2) Our method does not exhibit a significant performance decline compared to the base method.** Table 3 shows that pruning 25.0%, 33.3% and 40.0% of expert parameters reduces the model's average performance by 3.14%, 4.80% and 8.61%, respectively. These declines are acceptable, as dynamic pruning on quantized models inherently introduces additional performance trade-offs.

2.6 Analysis of Taylor-Linearized Approximation

This section provides a quantitative validation for the Taylor-linearized approximation used in Section 7 to accelerate expert importance evaluation. We randomly sample 100 prompts from MMLU and compute, at every layer l, two per-expert importance scores: the exact max-activation score $\sigma_{l,k}$ and its Taylor-linearized counterpart $\tilde{\sigma}_{l,k}$ for expert k. For a given m, let $S_l^{\text{exact}}(m)$ and $S_l^{\text{linear}}(m)$ denote the sets of Top-m experts ranked by $\sigma_{l,k}$ and $\tilde{\sigma}_{l,k}$, respectively. We then measure the Top-m set overlap ratio:

$$\text{Overlap}_l(m) = \frac{\left|S_l^{\text{exact}}(m) \cap S_l^{\text{linear}}(m)\right|}{m}, \tag{10}$$

which directly quantifies how many experts selected by the Taylor-linearized Top-m are also selected by the exact Top-m.

Figure **??** reports $\text{Overlap}_l(m)$ across layers under different compression ratios for both Mixtral 8×7B Instruct and Mixtral. We observe consistently high overlap across layers and settings, indicating that Taylor linearization recovers nearly the same Top-m experts as the exact max-activation metric. The results empirically support that the Taylor-linearized approximation preserves expert ranking quality while substantially reducing the computation required for expert scoring.

References

[1] Johnson J, Douze M, Jégou H. Billion-scale similarity search with GPUs. IEEE Transactions on Big Data, 2019, 7(3): 535–547

[2] Fan S, Jiang X, Li X, Meng X, Han P, Shang S, Sun A, Wang Y, Wang Z. Not all layers of llms are necessary during inference. arXiv preprint arXiv:2403.02181, 2024

[3] Zhang J, Juan D C, Rashtchian C, Ferng C S, Jiang H, Chen Y. Sled: Self logits evolution decoding for improving factuality in large language models. In: Globerson A, Mackey L, Belgrave D, Fan A, Paquet U, Tomczak J, Zhang C, eds, Advances in Neural Information Processing Systems. 2024, 5188–5209

[4] Chuang Y S, Xie Y, Luo H, Kim Y, Glass J R, He P. Dola: Decoding by contrasting layers improves factuality in large language models. In: The Twelfth International Conference on Learning Representations. 2024

[5] Fuglede B, Topsoe F. Jensen-shannon divergence and hilbert space embedding. In: International symposium onInformation theory, 2004. ISIT 2004. Proceedings. 2004, 31

[6] Kwon W, Li Z, Zhuang S, Sheng Y, Zheng L, Yu C H, Gonzalez J, Zhang H, Stoica I. Efficient memory management for large language model serving with pagedattention. In: Proceedings of the 29th Symposium on Operating Systems Principles. 2023, 611–626

[7] He Y, Fang J, Yu F R, Leung V C. Large language models (llms) inference offloading and resource allocation in cloud-edge computing: An active inference approach. IEEE Transactions on Mobile Computing, 2024

[8] Eliseev A, Mazur D. Fast inference of mixture-of-experts language models with offloading. arXiv preprint arXiv:2312.17238, 2023

[9] Wang Y, Peng Q, Liu H, Xu H, Shao M, Wang W. Deep expertise and interest personalized transformer for expert finding. Information Processing & Management, 2024, 61(5): 103773

[10] Shi X H, Liang Y C, Lee H P, Lu C, Wang Q. Particle swarm optimization-based algorithms for tsp and generalized tsp. Information processing letters, 2007, 103(5): 169–176

[11] Shi W, Min S, Lomeli M, Zhou C, Li M, Lin X V, Smith N A, Zettlemoyer L, Yih t W, Lewis M. In-context pretraining: Language modeling beyond document boundaries. In: The Twelfth International Conference on Learning Representations. 2024

[12] Clark P, Cowhey I, Etzioni O, Khot T, Sabharwal A, Schoenick C, Tafjord O. Think you have solved question answering? try arc, the ai2 reasoning challenge. arXiv preprint arXiv:1803.05457, 2018

[13] Clark C, Lee K, Chang M W, Kwiatkowski T, Collins M, Toutanova K. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In: Burstein J, Doran C, Solorio T, eds, 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). June 2019, 2924–2936

[14] Zellers R, Holtzman A, Bisk Y, Farhadi A, Choi Y. HellaSwag: Can a machine really finish your sentence? In: Korhonen A, Traum D, Màrquez L, eds, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. July 2019, 4791–4800

[15] Hendrycks D, Burns C, Basart S, Zou A, Mazeika M, Song D,

Steinhardt J. Measuring massive multitask language understanding. Proceedings of the International Conference on Learning Representations (ICLR), 2021
[16] Sakaguchi K, Bras R L, Bhagavatula C, Choi Y. Winogrande: an adversarial winograd schema challenge at scale. Commun. ACM, 2021, 64(9): 99–106
[17] Contributors O. Opencompass: A universal evaluation platform for foundation models, 2023
[18] Merity S, Xiong C, Bradbury J, Socher R. Pointer sentinel mixture models. arXiv preprint arXiv:1609.07843, 2016
[19] Raffel C, Shazeer N, Roberts A, Lee K, Narang S, Matena M, Zhou Y, Li W, Liu P J. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 2020, 21(1)
[20] Li P, Jin X, Cheng Y, Chen T. Examining post-training quantization for mixture-of-experts: A benchmark. arXiv preprint arXiv:2406.08155, 2024
[21] Lu X, Liu Q, Xu Y, Zhou A, Huang S, Zhang B, Yan J, Li H. Not all experts are equal: Efficient expert pruning and skipping for mixture-of-experts large language models. In: Ku L W, Martins A, Srikumar V, eds, Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). August 2024, 6159–6172
[22] Liu E, Zhu J, Lin Z, Ning X, Blaschko M B, Yan S, Dai G, Yang H, Wang Y. Efficient expert pruning for sparse mixture-of-experts language models: Enhancing performance and reducing inference costs. arXiv preprint arXiv:2407.00945, 2024
[23] Huang W, Liao Y, Liu J, He R, Tan H, Zhang S, Li H, Liu S, Qi X. Mixture compressor for mixture-of-experts llms gains more. In: The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025. 2025
[24] Jiang A Q, Sablayrolles A, Roux A, Mensch A, Savary B, Bamford C, Chaplot D S, Casas D d l, Hanna E B, Bressand F, others . Mixtral of experts. arXiv preprint arXiv:2401.04088, 2024
[25] Liu A, Feng B, Wang B, Wang B, Liu B, Zhao C, Dengr C, Ruan C, Dai D, Guo D, others . Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. arXiv preprint arXiv:2405.04434, 2024
[26] Yadavilli V S, Seshadri K, S. N B. Joint modeling of causal phrases-sentiments-aspects using hierarchical pitman yor process. Information Processing & Management, 2024, 61(4): 103753
[27] Liu J, Yi B, Zhang H, Shen X, Song L, Lei Y, Zheng H. Modeling semantic representation with llm-enhanced for knowledge-aware recommendation. Information Processing & Management, 2026, 63(2, Part A): 104387
[28] Tarpey T. Linear transformations and the k-means clustering algorithm: applications to clustering curves. the american statistician, 2007, 61(1): 34–40
[29] Douze M, Guzhva A, Deng C, Johnson J, Szilvasy G, Mazaré P E, Lomeli M, Hosseini L, Jégou H. The faiss library. arXiv preprint arXiv:2401.08281, 2024