FIRM-MoE: Fine-Grained Expert Decomposition for Resource-Adaptive MoE Inference

Anonymous submission

Abstract

Mixture-of-Experts (MoE) is a sparse neural architecture that significantly increases model capacity while maintaining low computational complexity. However, deploying MoE-based large language models (LLMs) on memory-constrained edge devices remains challenging due to their substantial memory requirements. To address this issue, we propose *FIRM-MoE*, a fine-grained expert offloading framework designed to enable flexible and efficient MoE inference. The core insight of our approach is to reduce the risk of inaccurate expert loading by decomposing each expert into fine-grained sub-experts and then dynamically allocating them through a fine-grained scheduling strategy. To further reduce the error in expert loading, we introduce a multi-layer expert prediction mechanism and a resource-adaptive expert pre-loading algorithm to enable more robust expert allocation. This design allows our model to achieve more efficient expert utilization and improved resilience to prediction errors. We conduct extensive experiments to demonstrate the superiority of *FIRM-MoE* across diverse memory constraints. The results show that *FIRM-MoE* achieves up to 1.5× speedup and 2.8× memory savings in decoding, compared to state-of-the-art MoE offloading strategies.

Introduction

Large language models (LLMs) have achieved state-of-the-art performance across various domains, including generative natural language processing (Achiam et al. 2023) and complex reasoning (Guo et al. 2025). Despite their promise, the parameter scale of LLMs—often exceeding 500 billion—poses substantial obstacles to deployment on edge devices with limited computational resources.

To mitigate computational overhead while preserving model capacity, Mixture-of-Experts (MoE) architectures (Shazeer et al. 2017) sparsify LLMs by adaptively activating only a subset of parameters within a large parameter pool. However, this efficiency comes at a cost: MoE-based LLMs require substantial memory to accommodate their large expert pools. A natural approach is to prune (Kim et al. 2024) or quantize (Frantar et al. 2022) the experts, though this may degrade their representational power. Other methods leverage on-demand expert loading, but as shown in Figure 1(a), these approaches often suffer from low resource utilization and high latency. To address this, several recent works (Eliseev and Mazur 2023; Zhong et al. 2024;

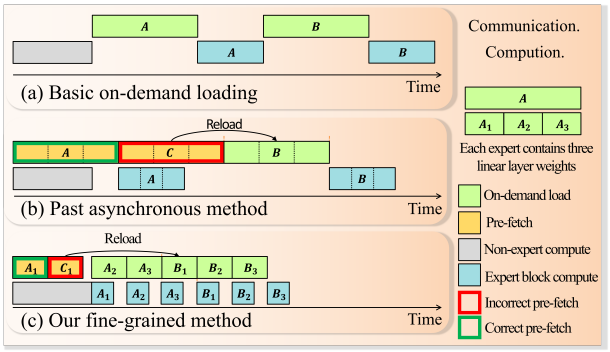

Figure 1: Example fine-grained pipeline parallel schedule. Demonstrates the basic paradigm of MoE LLMs in edge inference and the optimization in this paper when the GPU is unable to load the complete model due to memory limitations.

Tang et al. 2024) propose pre-fetching and predicting future experts to enable compute-communication overlap and improve utilization. However, as illustrated in Figure 1(b), such methods heavily depends on the accuracy of expert prediction. Incorrect pre-loading can result in resource misallocation and even increased latency due to high CPU-to-GPU data transfer overhead. Moreover, these approaches (Eliseev and Mazur 2023; Xue et al. 2024; Tang et al. 2024) often adopt fixed configurations, lacking adaptability to real-world dynamic workloads.

To tackle these limitations, this paper introduces *FIRM-MoE*, a generalized and resource-adaptable framework for MoE-style LLMs. First, to address loading latency, we introduce a granular decomposition strategy that decomposes MoE experts into finer-grained sub-experts (Figure 1(c)). This enables on-demand activation of only the necessary components during pre-fetching, effectively minimizing memory waste caused by the transmission of inactive experts. Next, we propose Meeting-of-Layers (MoL), a multi-layer collaborative prediction mechanism that aggregates cross-layer signals to enhance the effectiveness of expert selection and mitigate the impact of misprediction. Lastly, we present Hierarchical Expert Optimization and Pre-fetching (HEOP), a resource-adaptive algorithm designed to dynamically adjust pre-loading strategies across model layers. By doing so, HEOP achieves efficient inference with reduced

latency, even under stringent resource constraints. Our main contributions are summarized as follows:

- We propose a Fine-grained expert decomposition by decomposing experts into sub-experts, a simple yet underexplored approach in prior MoE literature.
- A multi-layer collaborative prediction mechanism is introduced to improve expert pre-loading accuracy through cross-layer interactions.
- We propose a hierarchical and resource-adaptive expert pre-fetching algorithm. It dynamically adapts to GPU memory availability to achieve optimal inference performance.
- We evaluate *FIRM-MoE* on representative models from the Qwen, DeepSeek, and OLMoE series of MoE architectures using an RTX 3090 24GB GPU. Compared to state-of-the-art baselines, our method achieves up to 1.5× inference speedup and 2.8× memory savings.

Related Works

Mixture-of-Experts

Mixture of Experts was proposed in earlier research (Jacobs et al. 1991). Its core structure consists of a set of "expert networks" and a gating network. Each expert is usually responsible for handling one sub-task in a specific field, and the gating network dynamically routes input data to the most relevant expert. Sparse gating (Shazeer et al. 2017) was proposed to only activates a subset of experts for each round of forward calculation, reducing the computational overhead of the overall model. In recent years, the MoE architecture has been widely integrated into LLMs, achieving a balance between parameter expansion and inference efficiency. (Fedus, Zoph, and Shazeer 2022; Wei et al. 2024)

Efficient Inference on MoE-style LLMs

With the rapid development of MoE-LLMs, efficiently deploying MoE models for inference on local devices has become a prominent research focus. Many inference methods and systems have been developed. Adaptive-MoE (Li et al. 2023) allows tokens to be processed by varying numbers of experts, which improves computational efficiency but inevitably leads to performance degradation. Pre-gated MoE (Hwang et al. 2024) improves activation accuracy by introducing additional predictive gates; however, structural modifications compromise its generalization ability in practical scenarios. Some studies (Kamahori et al. 2024; Cao et al. 2025) have attempted to leverage CPU resources for collaborative inference. Fast-MoE (Eliseev and Mazur 2023) proposed using hidden states from the previous layer to predict expert activation in the next layer. Hobbit (Tang et al. 2024) further validated the approach from the perspective of inter-layer activation similarity. MoE-Infinity (Xue et al. 2024) proposed a statistical method for activation path prediction, which does not require model modifications but suffers from limited prediction accuracy and high computational overhead. These methods generally overlook the potential of expert splitting for finer-grained control. Although Adap-MoE (Zhong et al. 2024) attempts to make computation and loading more fine-grained, it remains a single-point optimization. In contrast, our framework integrates expert decomposition into both the loading and scheduling stages, yielding system-wide benefits.

Method

FIRM-MoE consists of three main parts, including Expert Decomposition, Meeting-of-Layers (MoL), and the Hierarchical Expert Optimization and Pre-fetching (HEOP).

Overview of *FIRM-MoE*

Previous methods (Zhong et al. 2024; Tang et al. 2024) scheduled experts as whole units, leading to inefficient resource use. We address this by introducing a fine-grained scheduling framework that decomposes each expert into smaller linear-layer components, enabling precise coordination of computation and memory during execution, as illustrated in Figure 2.

Specifically, our fine-grained scheduling framework consists of a *Pre-fetch Manager*, and an *Expert Manager*. The *Pre-fetch Manager* predicts the activated experts in the next layer and triggers experts pre-loading once prediction is complete. *Expert Manager* is responsible for scheduling both the expert pre-loading and on-demand loading workloads, as well as the expert computation.

To mitigate resource waste caused by inaccurate predictions, we design the MoL algorithm within the *Pre-fetch Manager*. Furthermore, to improve efficiency under different resource constraints, we integrate the resource-adaptive HEOP algorithm into the *Expert Manager*. We also deploy a cache pool on the GPU for storing transmitted expert submodules. Following the design in prior studies (Xue et al. 2024; Tang et al. 2024; Yi et al. 2025), an LRU caching strategy is implemented to retain the most frequently activated experts. The entire fine-grained scheduling system adopts a multi-threaded architecture, enabling precise asynchronous execution of expert computation and loading.

Expert Decomposition and Prediction

A typical MoE layer comprises N expert modules and a routing network. Given an input $\mathbf{x}_l$ at layer $l \in \{1, \ldots, L\}$, the router selects the top-k experts and distributes the input accordingly. **Spare MoE layers.** The router at layer l computes expert selection scores as follows:

$$G_l(x) = \text{Softmax}\left(\text{TopK}(\mathbf{x}_l \cdot W_g^l, k)\right), \tag{1}$$

where $W_g^{(l)} \in R^{d_{\text{in}} \times N}$ is the routing weight matrix that maps the input to expert logits, and $\text{TopK}(\cdot, k)$ retains the k highest scores, setting the rest to zero. The softmax function normalizes the selected logits to yield the gating weights $G_l(\mathbf{x})$.

The output of the MoE layer is a weighted sum over the outputs of the selected experts:

$$y_l = \sum_{i=1}^{N} G_l(\mathbf{x}_l)_i \cdot E_i(\mathbf{x}_l), \tag{2}$$

where $E_i(\cdot)$ denotes the output of the i-th expert.

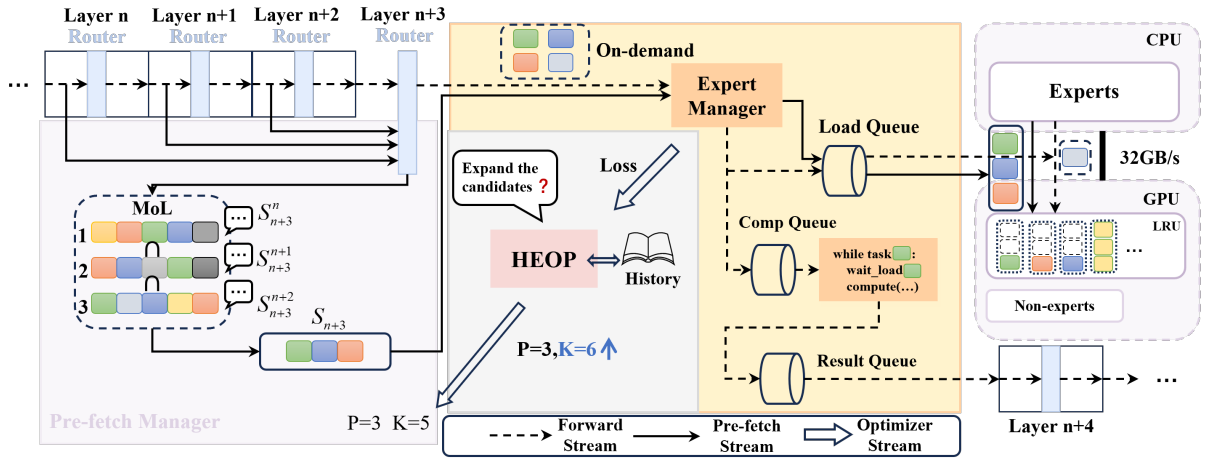

Figure 2: **Overview of *FIRM-MoE*.** The left shows MoL, where predictions are collaboratively generated by preceding layers. The middle illustrates the HEOP execution flow and fine-grained expert transmission and computation queues, where computation starts once each expert weight finishes loading. The right shows the hierarchical CPU–GPU expert memory structure.

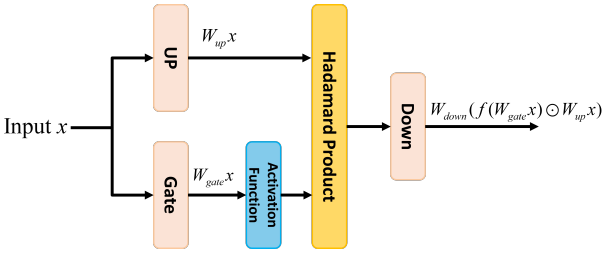

Figure 3: An overview of expert architecture in the DeepSeek, Qwen, and OLMoE series of MoE models, containing three main weight matrices: W_{up}, W_{gate}, and W_{down}.

Expert Decomposition. As shown in figure 3, each expert typically consists of two (Jiang et al. 2024) or three linear transformation matrices. For instance, in the Qwen-series MoE models, each expert E_i contains three matrices: W_{gate}, W_{up}, and W_{down}. The forward computation within an expert is defined as:

$$E_i(\mathbf{x}) = W_{\text{down}}\left(f\left(W_{\text{gate}}\mathbf{x}\right) \odot W_{\text{up}}\mathbf{x}\right), \tag{3}$$

where $f(\cdot)$ is an element-wise activation function (e.g., `SiLU`), and $\odot$ denotes Hadamard (element-wise) product. We treat each weight matrix (e.g., W_{gate}, W_{up}, W_{down}) as a decomposable and independently loadable sub-experts during expert management and transmission.

Expert Prediction and Pre-fetching. Before MoE inference, expert prediction is first performed by feeding the routing input from the previous layer $\mathbf{x}_l$ to MoE layer. Assume the experts potentially activated at layer $l+1$ are predicted based on x_l, the predicted sets of experts S_{l+1} can be expressed as follows:

$$\mathcal{S}_{l+1} = \text{TopKIndices}(\mathbf{x}_l \cdot W_g^{(l+1)}, k). \tag{4}$$

where $\text{TopKIndices}(\cdot)$ denotes the indices of the *topk* scoring experts. $W_g^{(l+1)}$ represents the weight matrix of the router at the $(l+1)$ layer, and x_l is the input to the router at layer l.

After obtaining the predicted experts set, the corresponding experts are pre-loaded to the GPU for performing computation. Depending on the model scale, different pre-fetching granularities (n) can be applied to balance loading overhead. A detailed analysis of this design is presented in the experimental section under Fine-Grained Pre-fetching.

Meeting-of-Layers (MoL)

Current expert prediction approaches rely on adjacent layer features and suffer from limited accuracy, particularly in the shallow layers of the model (Zhong et al. 2024).

In *FIRM-MoE*, we introduce MoL mechanism within the *Pre-fetch Manager* (Figure 2). *Pre-fetch Manager* coordinates the layers participated in expert prediction, then selects the most favored expert from among the numerous candidates.

Specifically, in the prediction method based on Eq. (4), when predicting the experts for layer l, the *Pre-fetch Manager* jointly involves the preceding P layers $(P<l)$ in the decision-making process. Each participating layer $m \in [l-P, l)$ independently predicts K candidate experts, with the prediction function formally defined as follows:

$$S_l^{(m)} = \text{TopKIndices}(x_m \cdot W_g^{(l)}, K), \tag{5}$$

where $S_l^{(m)}$ denotes the prediction result of layer m for the experts in layer l. Then, from all predicted candidates, high-confidence experts are selected to form the predicted expert set S_l for layer l:

$$S_l = \bigcap_{m=l-P}^{l-1} S_l^{(m)}, \tag{6}$$

Compared to simply expanding the prediction range or taking the union of candidate sets, selecting high-confidence

predictions more effectively leverages inter-layer similarity to improve prediction quality.

Evaluation Metric. *Prediction Accuracy* is defined as the proportion of activated experts that are correctly predicted (Eliseev and Mazur 2023):

$$\textit{Prediction Accuracy} = \frac{S_l \cap A_l}{|A_l|}, \tag{7}$$

where A_l is the total number of experts activated at layer l. *Prediction Accuracy* favors high prediction coverage rate, ignoring the memory and CPU-GPU I/O overhead introduced by pre-fetching excessive, unused experts.

Expert Utilization: To address the above issue, we define a new metric named *Expert Utilization* that measures the proportion of experts that are actually activated within the prediction set:

$$\textit{Expert Utilization} = \frac{S_l \cap A_l}{|S_l|}. \tag{8}$$

Expert Utilization favors high expert prediction precision. For edge deployment with resource constraints, this metric ensures minimum overhead for expert pre-fetching. The experimental section provides a detailed comparison of *Expert Utilization* given different prediction strategies.

Resource-Adaptive Dynamic Pre-fetching

We design the MoL algorithm to predict the set of experts involved in computation. Building upon this, we propose HEOP, a resource-adaptive dynamic pre-fetching algorithm to enhance MoE inference efficiency under strict computational resource constraints.

Optimization Objective. In *FIRM-MoE*, we aim to minimize the processing latency of MoE model inference, and formulate the following optimization problem:

$$K, P = \arg\min_{K,P} \quad T(K, P) \tag{9}$$

$$\text{s.t.} \quad C \leq C_{max} \tag{10}$$

where $T(K, P)$ is the inference latency, K is the number of candidate experts per layer, and P is the number of layers participating in expert prediction. C denotes memory usage, constrained by the maximum available memory C_{max}.

Problem Complexity. The joint search over layer index, prefetch depth P, and number of candidate experts K forms a cubic search space, yielding a worst-case complexity of $\mathcal{O}(n^3)$. For example, in Qwen1.5-MoE-A2.7B with 24 layers, this results in tens of thousands of simulated inferences—making exhaustive search impractical.

Hierarchical Expert Optimization and Pre-fetching (HEOP). We propose HEOP for searching optimal configurations that minimizes the optimization goals in Eq. (9).

Specifically, we defined the following loss function for HEOP:

$$\mathcal{L}_T = \alpha \cdot n_{unused} + \beta \cdot (n_{miss} - n_{hit}) \tag{11}$$

The loss function evaluates MoL performance by balancing over- and under-prefetching. The α term penalizes unused experts that were prefetched but not used. The β term penalizes missing experts—those needed but not prefetched—while accounting for cache hits. Formally, the β term is $(n_{\text{miss}} - n_{\text{hit}})$, where n_{miss} counts required experts absent from prefetching, and n_{hit} counts those already in the GPU cache.

Algorithm 1: Hierarchical Expert Optimization and Pre-fetching (HEOP)

Input: default (K, P), weights (α, β), interval N_{update}, cache limit $C_{\max}$, model M, patience Pat
Output: group-wise optimal (K_g^*, P_g^*)
`// Initialize`
$\mathcal{G} = \{g_{\text{shallow}}, g_{\text{middle}}, g_{\text{deep}}\} \leftarrow M$
history $H_g \leftarrow \emptyset$, cost $L_T^g \leftarrow \emptyset$, $\forall g \in \mathcal{G}$
Direction$_g \leftarrow$ rand($\{\text{K}, \text{P}\}$), $\delta_g \leftarrow$ rand($\{-1, +1\}$)
`// Start Optimization`
for each token step t **do**
  for each $g \in \mathcal{G}$ **do**
    Record $n_{\text{unused}}, n_{\text{miss}}, n_{\text{hit}}$
    $\mathcal{L}_T^g = \alpha \cdot n_{\text{unused}} + \beta \cdot (n_{\text{miss}} - n_{\text{hit}})$
    $H_g \leftarrow H_g \cup \{\mathcal{L}_T^g\}$
  end for
  if $t \bmod N_{\text{update}} = 0$ **then**
    for each $g \in \mathcal{G}$ **do**
      $\bar{\mathcal{L}}^g \leftarrow \text{avg}(H_g)$
      if no improvement in past Pat steps **then**
        $\delta_g \leftarrow -\delta_g$ // Switch Direction$_g$
      end if
      if Direction$_g$ = K **then**
        $K_g \leftarrow K_g + \delta_g$
      else
        $P_g \leftarrow P_g + \delta_g$
      end if
      Apply (K_g, P_g) to all layers in g
    end for
  end if
end for`// Optimization completed`
for each $g \in \mathcal{G}$ **do**
  $(K_g^*, P_g^*) \leftarrow \arg\min_{(K,P)} \bar{\mathcal{L}}_g$
  Apply (K_g^*, P_g^*) to group g
end for

Optimization Process. Observing that expert pre-fetching accuracy differs between shallow and deep layers (Tang et al. 2024), we partition the model layers into three groups: shallow, middle, and deep. For each group, we apply a coordinate-wise hill climbing strategy to optimize expert configurations. By optimizing groups independently, the search complexity is reduced to $\mathcal{O}(n)$, alleviating the scalability challenges posed by the large number of experts.

As shown in Algorithm 1, HEOP maintains a group-specific prefetch configuration (K_g, P_g) for each group g. At each token step, the system records the predicted experts, the actually activated experts, and the cache-hit experts, and computes the group-wise cost $\mathcal{L}_T^g$ according to Eq. (11). Every N_{update} steps, the algorithm performs a lightweight directional search: either K_g or P_g is adjusted by a small step $\delta_g \in -1, +1$ based on recent cost trends. If no improvement is observed over Pat steps, the search direction δ_g is reversed and the search dimension is switched between K

and P. After the profiling phase, the configuration (K_g^*, P_g^*) with the lowest historical average cost is selected and fixed for each group.

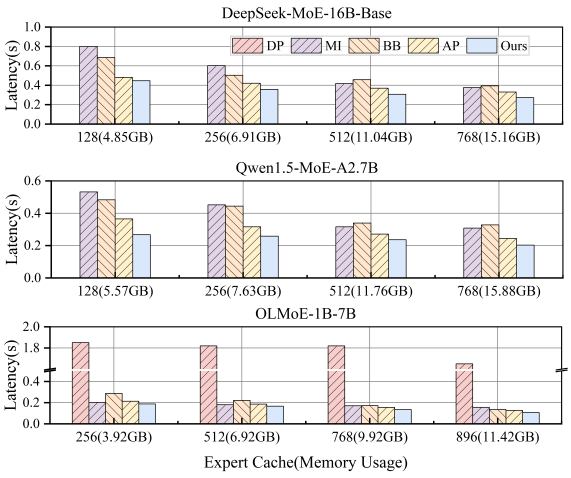

Figure 4: Inference latency under various GPU memory constraints.

Experiments

Experiments Setup

Model and Datasets. To evaluate the performance of *FIRM-MoE*, we test five models from Hugging Face (Face 2025) with varying sizes: Qwen1.5-MoE-A2.7B(14B-A2.7B) (Team 2024), Qwen3-30B-A3B(30B-A3B) (Team 2025), DeepSeek-MoE-16B-Base(16B-A6.5B)(Dai et al. 2024), DeepSeek-V2-Lite(16B-A2.4B) (DeepSeek-AI 2024), and OLMoE-1B-7B(7B-A7.3B) (Muennighoff et al. 2024). Following previous work (Tang et al. 2024; Kamahori et al. 2024), we evaluate our method using the TruthfulQA (Lin, Hilton, and Evans 2021) dataset for question-answering generation. Since the routing behavior of MoE models can be significantly influenced by the characteristics of the input data distribution, we further conduct quantitative evaluation on the ShareGPT (Anon 2023) dataset to measure the end-to-end performance of the model under varying input and output lengths.

Hardware. To evaluate the performance of our *FIRM-MoE* in an edge environment, we deployed it on a device equipped with an NVIDIA GeForce RTX 3090 GPU (24 GB VRAM) and a 32-core CPU with 64 GB of memory. The CPU and GPU are interconnected via a PCIe 4.0 bus with a theoretical bandwidth of 32 GB/s.

Metrics. Given that edge service scenarios primarily involve the generation process of LLMs, we use decoding speed as our performance metric. For evaluation, we select 50 high-quality samples with a maximum output length of 1024 tokens each. We measure decoding latency per output token, averaged across all samples.

Baseline and Implementation Details. We compare our method with several SOTA inference strategies: 1) AP (Zhong et al. 2024), which uses a similarity-based prediction method with a focus on cache design; 2) MI (Xue et al. 2024) predicts expert similarity based on statistical information from activation paths. We implement MI with 100 and matrix clusters for different performance. 3) DeepSpeed-MII (Microsoft 2023) adopts ZeRO-Infinity (Rajbhandari et al. 2021) to offload model parameters to CPU and load them back on demand during inference. We also enable `pin_memory` to improve CPU memory read/write performance and reduce fragmentation. 4) Fiddler (Kamahori et al. 2024) is a method that leverages CPU-GPU collaborative computation. 5) Llama.cpp (ggml authors 2023), together with Fiddler, is used as a baseline to quantitatively analyze the impact of varying input and output lengths. For Llama.cpp, we configure the `ngl` parameter—which controls the number of layers executed on the GPU—based on the available expert cache size, ensuring a fair comparison under different memory constraints. 6) We implement basic equal-ratio expert pre-fetching and the coarse-grained pre-loading strategy proposed in (Eliseev and Mazur 2023) as our baseline (BB).

Following prior work (Eliseev and Mazur 2023; Hwang et al. 2024; Kong et al. 2023), we fix the batch size to 1. We set 5 different expert cache size limits: 128, 256, 512, 768, and 896. Notably, our approach is orthogonal to model compression techniques such as expert quantization, thereby preserves model accuracy.

Experiments Results

Figure 5 shows the end-to-end performance of our method across various models and input/output length settings. *FIRM-MoE* consistently outperforms Fiddler and Llama.cpp, achieving an average 1.31× throughput gain. Fiddler suffers from noticeable latency under multi-expert models due to its brute-force search strategy for balancing GPU-CPU workloads. For fairness, we replaced it with a greedy search variant in our evaluation, yet its performance remained suboptimal.

Figure 4 presents inference latency under different GPU expert cache sizes. Across all settings, *FIRM-MoE* achieves lower latency than all baselines. On the Qwen1.5-MoE-A2.7B model, where expert transfers dominate due to limited cache size (128 experts), *FIRM-MoE* achieves up to 1.8× speedup over BB; even with 768 experts, it maintains a 1.6× advantage. Notably, when comparing the latency between 128 and 768 expert caches, *FIRM-MoE* incurs less than 30% performance degradation while achieving a 2.85× reduction in memory usage. This significant memory saving makes *FIRM-MoE* particularly suitable for deployment on resource-constrained edge devices. As for DeepSpeed-MII (DP), since it does not support fine-grained control over expert caching during inference, we approximate its behavior by proportionally retaining a fixed amount of parameters on the GPU. However, this design leads to the frequent transfer of entire expert layers for each MoE layer, introducing substantial overhead and resulting in degraded performance. Therefore, we only report its latency on the relatively small OLMoE-1B-7B model.

The results demonstrate that *FIRM-MoE* consistently delivers superior performance under both varying input/output lengths and different cache constraints. They also reveal

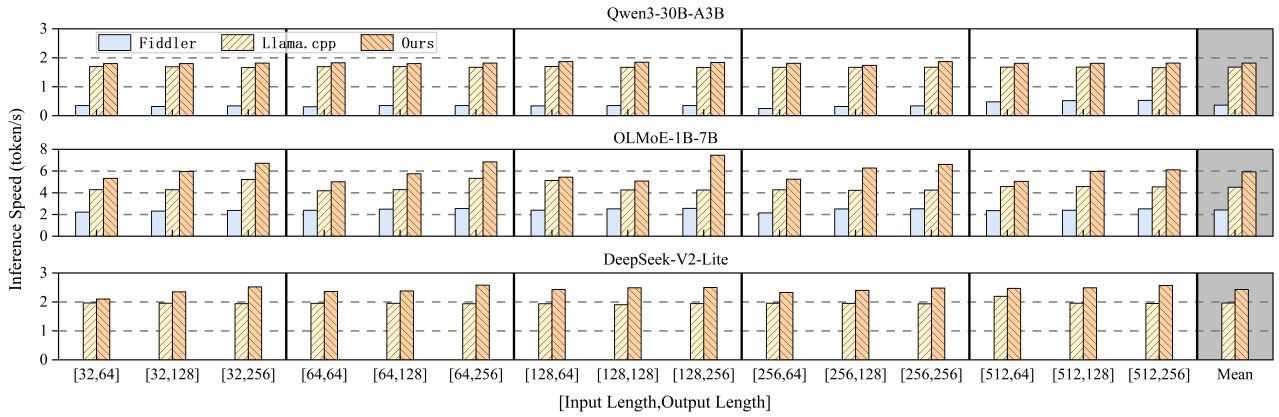

Figure 5: End-to-end performance comparison across three models of different sizes, measured by the number of tokens generated per second (higher is better), under 15 different input/output length configurations. The rightmost group of bars represents the average performance over all 15 configurations.

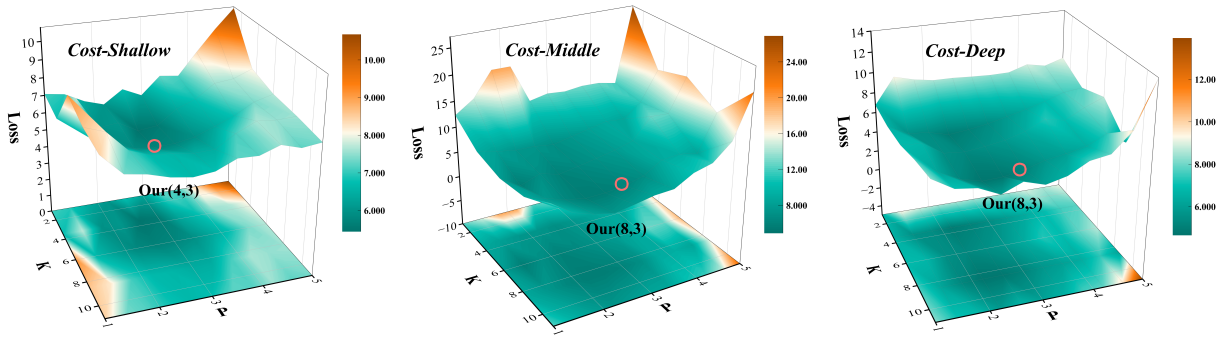

Figure 6: Distribution of cost across different (K, P) configurations on the OLMoE-1B-7B model under a cache size of 256. The first five layers are designated as shallow layers, the last four as deep layers, and the remaining as middle layers.

that under limited cache conditions, system performance becomes highly sensitive to prefetching accuracy. This observation will be further elaborated through the HEOP algorithm, and the trend is clearly illustrated in Figure 6 and discussed in subsequent sections.

HEOP Results

We conduct a detailed performance analysis of HEOP under low-resource conditions, focusing on how the hyperparameters K and P influence inference behavior. Using a balanced penalty configuration ($\alpha = 0.5$, $\beta = 0.5$) that preserves redundancy reduction benefits while mitigating expert under-utilization, Figure 6 illustrates the loss landscape for the OlMoE model under a 256-expert cache constraint, with darker hues corresponding to lower operational losses. The loss surface is obtained by averaging layer-wise computational losses (shallow, intermediate, and deep layers) across large-scale inference tasks. Loss measurements are sampled at 10-token intervals and aggregated through sequence-level averaging.

From the figure, we observe that the number of layers participating in expert prediction varies across groups. In shallow layers, where prediction accuracy fluctuates, the HEOP adopts a conservative strategy with $\{K = 4, P = 3\}$. For middle and deep layers with more stable prediction patterns, it favors larger candidate sets $\{K = 8, P = 3\}$. Comparing the left and middle subfigures in Figure 6, we find that low-loss regions in shallow layers generally occur at smaller K values, leading HEOP to narrow its prediction range. In contrast, for middle and deep layers, HEOP tends to select larger K values ($K > 5$). This layer-aware strategy effectively captures distinct loss characteristics across different depths, enabling HEOP to assign layer-specific optimal configurations.

Ablation Studies

To validate the effectiveness of each component in *FIRM-MoE*, we conduct ablation studies on the fine-grained prefetching technique, and the MoL algorithm.

Impact of Fine-Grained Pre-fetching. To evaluate the benefits of fine-grained expert pre-fetching, we modified the baseline method to pre-fetch only the first linear layer of the predicted experts ($n = 1$). We conducted inference on approximately 2,000 tokens from the TruthfulQA dataset and report the average results. As shown in Table 1, the fine-grained approach achieves speedups ranging from 1.24× to 1.41× under different cache constraints. This indicates that fine-grained pre-fetching effectively mitigates the penalty of

Method	Cache=128		Cache=256		Cache=512		Cache=768	
	Latency↓	Speedup↑	Latency↓	Speedup↑	Latency↓	Speedup↑	Latency↓	Speedup↑
Baseline	0.483	–	0.452	–	0.339	–	0.328	–
+ FG	0.365	1.32×	0.345	1.31×	0.273	1.24×	0.233	1.41×
+ MoL	0.400	1.21×	0.370	1.22×	0.281	1.21×	0.259	1.27×
+ FG + MoL	0.337	1.43×	0.317	1.43×	0.284	1.19×	0.230	1.43×
All	**0.292**	**1.65×**	**0.277**	**1.62×**	**0.263**	**1.29×**	**0.212**	**1.54×**

Table 1: Performance of different design components in *FIRM-MoE* on the Qwen1.5-MoE-A2.7B model under different cache constraints. The baseline represents equal-ratio coarse-grained prefetching. FG denotes fine-grained prefetching, in which only the first linear layer of each expert is preloaded. MoL uses a configuration of $K = 3$ and $P = 3$.

inaccurate predictions. The results suggest that decomposing experts into finer-grained components for scheduling is particularly advantageous on edge devices where transmission is a major bottleneck.

To further investigate the impact of pre-fetching granularity, we evaluated two configurations with $n = 1$ and $n = 2$, while keeping the same K, P, settings across all models in our full system. The results are shown in Table 2. For OLMoE-1B-7B, we observe that $n = 1$ consistently outperforms $n = 2$. This is because HEOP tends to select a larger number of experts for pre-fetching in this smaller model, leaving limited room for asynchronous overlap between computation and transmission. Increasing the pre-fetching granularity to $n = 2$ results in additional transmission overhead, which in turn causes significant pipeline stalls during execution. In contrast, for Qwen1.5-MoE-A2.7B and DeepSeek-MoE-16B-Base, the $n = 2$ configuration generally achieves better performance. These larger models incur higher transmission costs per expert, leading HEOP to favor fewer but more accurate expert predictions. Thus, there remains sufficient asynchronous space between computation and data transfer, and using $n = 2$ better utilizes available transmission bandwidth without blocking the compute pipeline. Therefore, we set $n = 1$ for smaller models, and $n = 2$ for larger models to achieve better system efficiency.

Model	**n**	**Cache=256**	**512**	**768**	**896**
OlMoE	1	**5.31**	**6.04**	**7.44**	**9.4**
	2	4.76	5.7	6.7	8.72
Model	**n**	**Cache=128**	**256**	**512**	**768**
Qwen	1	3.51	3.6	3.8	4.71
	2	**3.75**	**3.87**	**4.32**	**4.92**
DeepSeek	1	**2.24**	2.73	2.86	3.47
	2	2.18	**2.8**	**3.26**	**3.67**

Table 2: The impact of fine-grained prefetching: $n = 1$ prefetches only the expert's first linear layer, while $n = 2$ prefetches the first two. Model end-to-end throughput is reported for both settings (tokens/s).

Impact of MoL. Figure 7 compares expert utilization and prediction accuracy across prediction strategies. Our MoL approach achieves the highest expert utilization (90.5%), a 1.14× gain over Top4 (78.8%), but with lower prediction accuracy (59.8%). This trade-off stems from HEOP 's design, which favors high utilization to suit resource-constrained edge systems, thereby reducing unnecessary CPU–GPU transfers. The union strategy significantly enlarges the candidate set, but its low prediction precision leads to an increase in false positives and thus lower expert utilization. Weighted Voting can match our utilization only at the strictest threshold ($\geq$3.0), but this requires extra hyper-parameter tuning. In contrast, MoL requires no extra parameters, and is simple to implement.

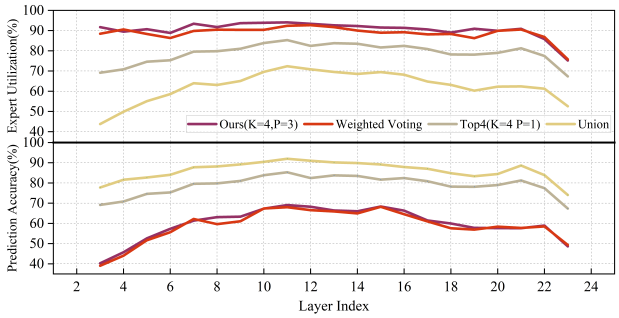

Figure 7: Expert utilization and prediction accuracy comparison on the Qwen1.5-MoE-A2.7B. Weighted Voting uses the same (K, P) but assigns layer-wise weights $[1.5, 1.0, 0.5]$ for the first three layers. *Top4* selects four experts per layer, matching the standard forward pass.*Union* merges all experts from $P = 3$ layers without filtering.

Conclusion

In this paper, we reveal that current works on MoE-style LLM inference pipelines face huge resource waste. To tackle this, We propose a general and highly adaptable MoE inference framework to accelerate single-GPU inference under varying memory constraints. We analyze the fundamental causes of inference inefficiency and introduce a fine-grained expert management strategy alongside a prediction refinement mechanism to improve accuracy. Furthermore, we design a search algorithm tailored to various expert caching scenarios to determine optimal inference configurations. Future work includes extending to a broader range of models and inference hardware to enhance practical applicability.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anon. 2023. ShareGPT Vicuna Unfiltered. https://huggingface.co/datasets/anon8231489123.

Cao, S.; Liu, S.; Griggs, T.; Schafhalter, P.; Liu, X.; Sheng, Y.; Gonzalez, J. E.; Zaharia, M.; and Stoica, I. 2025. Moe-lightning: High-throughput moe inference on memory-constrained gpus. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 715–730.

Dai, D.; Deng, C.; Zhao, C.; Xu, R.; Gao, H.; Chen, D.; Li, J.; Zeng, W.; Yu, X.; Wu, Y.; et al. 2024. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. *arXiv preprint arXiv:2401.06066*.

DeepSeek-AI. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434.

Eliseev, A.; and Mazur, D. 2023. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238*.

Face, H. 2025. Hugging face hub. https://huggingface.co/docs/hub/index.

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

Frantar, E.; Ashkboos, S.; Hoefler, T.; and Alistarh, D. 2022. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*.

ggml authors, T. 2023. llama.cpp. https://github.com/ggerganov/llama.cpp.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Hwang, R.; Wei, J.; Cao, S.; Hwang, C.; Tang, X.; Cao, T.; and Yang, M. 2024. Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 1018–1031. IEEE.

Jacobs, R. A.; Jordan, M. I.; Nowlan, S. J.; and Hinton, G. E. 1991. Adaptive mixtures of local experts. *Neural computation*, 3(1): 79–87.

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.; et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Kamahori, K.; Tang, T.; Gu, Y.; Zhu, K.; and Kasikci, B. 2024. Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models. *arXiv preprint arXiv:2402.07033*.

Kim, B.-K.; Kim, G.; Kim, T.-H.; Castells, T.; Choi, S.; Shin, J.; and Song, H.-K. 2024. Shortened llama: A simple depth pruning for large language models. *arXiv preprint arXiv:2402.02834*, 11.

Kong, R.; Li, Y.; Feng, Q.; Wang, W.; Ye, X.; Ouyang, Y.; Kong, L.; and Liu, Y. 2023. SwapMoE: Serving Off-the-shelf MoE-based Large Language Models with Tunable Memory Budget. *arXiv preprint arXiv:2308.15030*.

Li, J.; Su, Q.; Yang, Y.; Jiang, Y.; Wang, C.; and Xu, H. 2023. Adaptive gating in mixture-of-experts based language models. *arXiv preprint arXiv:2310.07188*.

Lin, S.; Hilton, J.; and Evans, O. 2021. TruthfulQA: Measuring How Models Mimic Human Falsehoods. arXiv:2109.07958.

Microsoft. 2023. Deepspeed-MII. https://github.com/microsoft/DeepSpeed-MII.

Muennighoff, N.; Soldaini, L.; Groeneveld, D.; Lo, K.; Morrison, J.; Min, S.; Shi, W.; Walsh, P.; Tafjord, O.; Lambert, N.; et al. 2024. Olmoe: Open mixture-of-experts language models. *arXiv preprint arXiv:2409.02060*.

Rajbhandari, S.; Ruwase, O.; Rasley, J.; Smith, S.; and He, Y. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 1–14.

Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; and Dean, J. 2017. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Tang, P.; Liu, J.; Hou, X.; Pu, Y.; Wang, J.; Heng, P.-A.; Li, C.; and Guo, M. 2024. Hobbit: A mixed precision expert offloading system for fast moe inference. *arXiv preprint arXiv:2411.01433*.

Team, Q. 2024. Qwen1.5-MoE: Matching 7B Model Performance with 1/3 Activated Parameters".

Team, Q. 2025. Qwen3 Technical Report. arXiv:2505.09388.

Wei, T.; Zhu, B.; Zhao, L.; Cheng, C.; Li, B.; Lü, W.; Cheng, P.; Zhang, J.; Zhang, X.; Zeng, L.; et al. 2024. Skywork-moe: A deep dive into training techniques for mixture-of-experts language models. *arXiv preprint arXiv:2406.06563*.

Xue, L.; Fu, Y.; Lu, Z.; Mai, L.; and Marina, M. 2024. MoE-Infinity: Offloading-Efficient MoE Model Serving. *arXiv preprint arXiv:2401.14361*.

Yi, R.; Guo, L.; Wei, S.; Zhou, A.; Wang, S.; and Xu, M. 2025. EdgeMoE: Empowering Sparse Large Language Models on Mobile Devices. *IEEE Transactions on Mobile Computing*.

Zhong, S.; Liang, L.; Wang, Y.; Wang, R.; Huang, R.; and Li, M. 2024. Adapmoe: Adaptive sensitivity-based expert gating and management for efficient moe inference. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 1–9.

Reproducibility Checklist

Instructions for Authors:
This document outlines key aspects for assessing reproducibility. Please provide your input by editing this `.tex` file directly.

For each question (that applies), replace the "Type your response here" text with your answer.

Example: If a question appears as

```
\question{Proofs of all novel claims
are included} {(yes/partial/no)}
Type your response here
```

you would change it to:

```
\question{Proofs of all novel claims
are included} {(yes/partial/no)}
yes
```

Please make sure to:

- Replace ONLY the "Type your response here" text and nothing else.
- Use one of the options listed for that question (e.g., **yes**, **no**, **partial**, or **NA**).
- **Not** modify any other part of the `\question` command or any other lines in this document.

You can `\input` this .tex file right before `\end{document}` of your main file or compile it as a stand-alone document. Check the instructions on your conference's website to see if you will be asked to provide this checklist with your paper or separately.

1. General Paper Structure

1.1. Includes a conceptual outline and/or pseudocode description of AI methods introduced (yes/partial/no/NA) yes

1.2. Clearly delineates statements that are opinions, hypothesis, and speculation from objective facts and results (yes/no) yes

1.3. Provides well-marked pedagogical references for less-familiar readers to gain background necessary to replicate the paper (yes/no) yes

2. Theoretical Contributions

2.1. Does this paper make theoretical contributions? (yes/no) no

If yes, please address the following points:

2.2. All assumptions and restrictions are stated clearly and formally (yes/partial/no) Type your response here

2.3. All novel claims are stated formally (e.g., in theorem statements) (yes/partial/no) Type your response here

2.4. Proofs of all novel claims are included (yes/partial/no) Type your response here

2.5. Proof sketches or intuitions are given for complex and/or novel results (yes/partial/no) Type your response here

2.6. Appropriate citations to theoretical tools used are given (yes/partial/no) Type your response here

2.7. All theoretical claims are demonstrated empirically to hold (yes/partial/no/NA) Type your response here

2.8. All experimental code used to eliminate or disprove claims is included (yes/no/NA) Type your response here

3. Dataset Usage

3.1. Does this paper rely on one or more datasets? (yes/no) yes

If yes, please address the following points:

3.2. A motivation is given for why the experiments are conducted on the selected datasets (yes/partial/no/NA) yes

3.3. All novel datasets introduced in this paper are included in a data appendix (yes/partial/no/NA) NA

3.4. All novel datasets introduced in this paper will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no/NA) NA

3.5. All datasets drawn from the existing literature (potentially including authors' own previously published work) are accompanied by appropriate citations (yes/no/NA) yes

3.6. All datasets drawn from the existing literature (potentially including authors' own previously published work) are publicly available (yes/partial/no/NA) yes

3.7. All datasets that are not publicly available are described in detail, with explanation why publicly available alternatives are not scientifically satisficing (yes/partial/no/NA) NA

4. Computational Experiments

4.1. Does this paper include computational experiments? (yes/no) yes

If yes, please address the following points:

4.2. This paper states the number and range of values tried per (hyper-) parameter during development of

the paper, along with the criterion used for selecting the final parameter setting (yes/partial/no/NA) yes

4.3. Any code required for pre-processing data is included in the appendix (yes/partial/no) no

4.4. All source code required for conducting and analyzing the experiments is included in a code appendix (yes/partial/no) no

4.5. All source code required for conducting and analyzing the experiments will be made publicly available upon publication of the paper with a license that allows free usage for research purposes (yes/partial/no) yes

4.6. All source code implementing new methods have comments detailing the implementation, with references to the paper where each step comes from (yes/partial/no) yes

4.7. If an algorithm depends on randomness, then the method used for setting seeds is described in a way sufficient to allow replication of results (yes/partial/no/NA) yes

4.8. This paper specifies the computing infrastructure used for running experiments (hardware and software), including GPU/CPU models; amount of memory; operating system; names and versions of relevant software libraries and frameworks (yes/partial/no) yes

4.9. This paper formally describes evaluation metrics used and explains the motivation for choosing these metrics (yes/partial/no) yes

4.10. This paper states the number of algorithm runs used to compute each reported result (yes/no) yes

4.11. Analysis of experiments goes beyond single-dimensional summaries of performance (e.g., average; median) to include measures of variation, confidence, or other distributional information (yes/no) no

4.12. The significance of any improvement or decrease in performance is judged using appropriate statistical tests (e.g., Wilcoxon signed-rank) (yes/partial/no) no

4.13. This paper lists all final (hyper-)parameters used for each model/algorithm in the paper's experiments (yes/partial/no/NA) yes