LITEMOE: MULTI-LAYER EXPERT PREDICTION AND TASK-AWARE ROUTING FOR MEMORY-EFFICIENT MOE INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing memory-efficient inference Mixture-of-Experts (MoE) systems offload experts to CPU memory and predict or preload experts one layer at a time. In multi-GPU and large-batch serving, these fine-grained transfers contend for PCIe bandwidth and expose expert-loading latency. We present LiteMoE, a memory-efficient MoE inference system built on two empirical properties of expert routing. First, expert activations across nearby layers are predictable enough to forecast the experts required by a group of future layers, allowing expert transfers to be coalesced and overlapped with computation. Second, tasks differ substantially in their sensitivity to routing errors, enabling selective expert loading when memory or transfer bandwidth is constrained. LiteMoE combines multi-layer expert prediction with task-aware expert selection based on routing sensitivity and predicted expert usage. Third, LiteMoE further uses task-aware scheduling to account for expert loading latency, reducing cross-request interference and SLO violations. Across multiple MoE models, workloads, batch sizes, and GPU configurations, LiteMoE improves throughput by up to 2.18× and reduces latency by up to 54% over state-of-the-art memory-efficient MoE inference systems while controlling output-quality degradation. When configured to preserve baseline accuracy, LiteMoE still achieves up to 1.85× higher throughput and 33% lower latency. These results show that exploiting cross-layer routing structure and task-dependent routing sensitivity can substantially reduce the communication cost of memory-efficient MoE inference.

1 INTRODUCTION

A recent trend in Deep learning (DL) has been enhancing the model performance by increasing the number of model parameters (Brown et al., 2020; Devlin et al., 2019). However, the linear relationship between the computational cost and the model size remains a crucial impediment to the practical applications of the large models. To circumvent this bottleneck, sparse architectures (Jacobs et al., 1991; Shen et al., 2020; Dong et al., 2019; 2020; Child et al., 2019; Lagunas et al., 2021) have been introduced. Among these, the Mixture-of-Experts (MoE) stands out as one of the most prominent sparse models. Its adoption has substantially amplified the scalability of Deep Neural Network (DNN) models across various DL applications.

An MoE layer comprises a gate and a collection of experts. The gate selects only a small number (e.g., 1 or 2) of experts for computation based on each input. Its sparse expert activation enables considerable expansion of model size by several orders of magnitude without a commensurate increase in computational requirements (FLOPs). However, the scaling of such models demands an excessively large amount of scarce GPU memory (Rajbhandari et al., 2022). MoE-based models incur significantly higher memory usage than their dense counterparts (Hwang et al., 2024; Rajbhandari et al., 2022), e.g., Mixtral can consume up to 14x more memory than its dense counterpart. Due to the intricate dependence between model capacity and memory requirements, designing a memory-optimized MoE inference system is crucial for serving very large models.

Prior works on MoE inference, such as (Xue et al., 2024; Yi et al., 2023; Huang et al., 2024; Kim et al., 2021; Shen et al., 2023), attempt to reduce the high memory footprint by offloading expert

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

parameters to CPU memory and loading them on an on-demand basis. While these strategies lower GPU memory usage during inference, they introduce significant latency due to the substantial overhead of transferring expert parameters from CPU to GPU. The recently proposed Pre-gatedMoE (Hwang et al., 2024) addresses this by predicting and preloading the experts needed for the next layer during the computation of the current layer, particularly in the single-GPU and single-batch-size scenario. It is effective in this scenario, as expert preloading time can be overlapped with and hidden by computation. However, as model sizes continue to scale, deploying a model across multiple GPUs (Eliseev & Mazur, 2023; Lepikhin et al., 2021; Jiang et al., 2024; Rajbhandari et al., 2022) has become increasingly common, and large batch sizes are often used to maximize throughput. Contention among multiple GPUs over limited PCIe bandwidth, combined with a longer expert loading time for a larger batch, prevents full overlap with computation, creating a performance bottleneck, as shown by our measurements (Section 2). Besides, in Pre-gatedMoE, expert predictions depend on the current layer's outputs, which introduces additional back-and-forth communication between computation and prediction. This leads to frequent transfer operations that incur substantial overheads, ultimately increasing end-to-end inference latency and decreasing throughput.

To address these limitations, we propose LiteMoE, a memory-efficient MoE inference framework that exploits structure in expert routing rather than treating each layer and request independently. Its design has three core components.

- **Multi-layer Expert Prediction.** It predicts and pre-fetches the required experts for several consecutive layers. This mitigates the bottleneck caused by layer-wise expert prediction and loading.
- **Task-aware Selective Expert Loading.** Our empirical analysis further reveals that, for certain tasks, MoE models can produce near-accurate outputs even when routed to imprecise or suboptimal experts. Building on this insight, LiteMoE selectively skips expert loading for tasks that are less sensitive to expert routing accuracy.
- **Task-aware Request Scheduling.** Unlike prior schedulers (Yu et al., 2022; Kwon et al., 2023; Agrawal et al., 2024; Zhong et al., 2024), LiteMoE explicitly incorporates expert loading latency into its scheduling decisions to minimize interference between concurrent requests in meeting their service-level-objectives (SLOs).

In summary, we make the following contributions:

- We perform empirical analyses to derive insights that motivate and guide our system design.
- We propose LiteMoE, which combines multi-layer expert prediction with task-aware selective expert loading to reduce CPU–GPU transfer frequency and exposed loading latency in multi-GPU and large-batch settings.
- We evaluate LiteMoE against state-of-the-art memory-efficient MoE inference systems. LiteMoE improves throughput by up to 2.18×, reduces latency by up to 54%, and increases accuracy by up to 13 percentage points, while limiting accuracy degradation relative to the ground truth to 3–9 percentage points. Even without any accuracy degradation, LiteMoE still achieves up to 1.85× higher throughput, 33% lower latency, and 17 percentage points higher accuracy.

2 Observation and Motivation

2.1 Experiment Settings

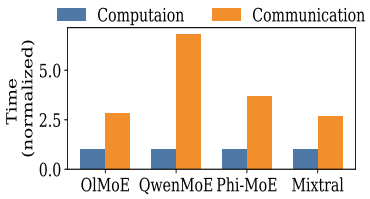


Figure 1: Expert loading time vs. expert compute time.

We performed empirical analyses on four publicly available MoE-based LLMs: OlMoE (Muennighoff et al., 2025), Qwen-1.5-MoE (Team, 2024) (QwenMoE), Phi-3.5-MoE-Instruct (Abdin et al., 2024) (Phi-MoE), and Mixtral-8x7B-Instruct-v0.1 (Jiang et al., 2024) (Mixtral) to uncover key bottlenecks and task-specific behaviors. We used an Intel Xeon processor equipped with 1.8 TB of host memory, and eight NVIDIA A100 Tensor Core GPUs, each with 40 GB of device memory. In the multi-GPU settings, tensor parallelism was employed. We used various popular natural language processing (NLP) tasks, including text summarization (SUM) (Zhang et al., 2020; Stiennon et al., 2020), question answering (QA) (Rajpurkar et al., 2016; Abacha, 2019), conversation (CONV) (OpenAssistant, 2023; Zheng et al., 2023), semantic similarity comparison (COMP) (Silcock et al., 2024; Cer et al.) and semantic classification (CLSFY) (Wallach, 2018; Bilal, 2026) . Based on (Yu et al., 2022) and (Kwon et al., 2023), the request arrival rate follows a Poisson distri-

bution. For OlMoE, Qwen-MoE, Phi-MoE, and Mixtral, the request rates were set to 0.8, 0.6, 0.5, and 0.5 req/s, respectively. For each time step, given the number of requests sampled from the distribution, individual inference requests are drawn from the dataset with uniform probability. We utilize vLLM (Kwon et al., 2023) as the inference engine. To evaluate accuracy, we treated the output of the full-expert MoE model as ground truth, following a methodology similar to (Liu et al., 2026). We obtained text embeddings for both the ground-truth and evaluated outputs using a pre-trained BERT model (Devlin et al., 2019) and measured their cosine similarity (Nikitin et al., 2024). Higher similarity indicates lower semantic deviation caused by incorrect token-to-expert routing.

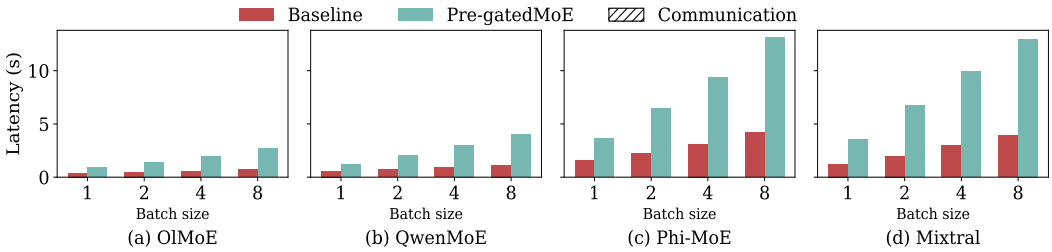


Figure 2: End-to-end inference latency.

2.2 Limitations of Layer-Wise Expert Prediction

To evaluate Pre-gatedMoE's behavior in practical serving environments, we performed experiments on our multi-GPU system. Figure 1 shows a comparison between average expert loading time and compute time across all layers. We observe that the communication time is approximately 2.7× to 6.8× greater than the computation time. Since the PCIe interconnect offers significantly lower bandwidth compared to GPU compute throughput, expert loading quickly becomes the dominant bottleneck as simultaneous expert transfers across GPUs amplify contention on PCIe links, leading to increased latency. Because expert loading time significantly exceeds computation time, it cannot be hidden or overlapped with computation. These results highlight a critical limitation of Pre-gatedMoE: while next-layer preloading may suffice for small-scale single-GPU inference, it fails to scale effectively. In multi-GPU and resource-constrained environments, expert loading becomes the primary bottleneck, severely limiting system efficiency and scalability.

Figure 2 illustrates how end-to-end latency grows across batch sizes, comparing baseline (which stores the entire MoE parameters in GPU memory) and Pre-gatedMoE. The hatched portion of the Pre-gatedMoE bars indicates expert loading time (i.e., communication time), reflecting the increasing GPU transfer cost with batch size. We observe that Pre-gatedMoE's latency rises by 2.1× to 3.7× compared to the baseline, primarily due to expert loading overhead.

Observation 1. In multi-GPU inference with large batch sizes, Pre-gatedMoE's per-layer preloading introduces substantial latency, motivating more efficient expert preloading strategies.

2.3 Tasks Vary in Sensitivity to Token-to-Expert Routing Accuracy

In this experiment, for a fixed fraction of early MoE layers, we route each token to previously loaded experts instead of its top experts. We then measure the cosine similarity between these outputs and those from the model with correct token routing in all MoE layers. Figure 3 plots cosine similarity versus the fraction of MoE layers with accurate routing. We see that for the classification and compare tasks, all models maintain above 90% similarity with respect to the full model even when all tokens are routed incorrectly. We also note that open-ended tasks such as conversation and summarization show low tolerance to inaccurate expert routing. Even with 75% accurate MoE layers, the similarity score falls below 80%. On the other hand, for the QA tasks which need to be more specific as the answer is in the context, the similarity score is above 80% for all models when only 50% of the MoE layers have accurate routing. For the summarization tasks, the accuracy falls below 80% when 50% of the MoE layers are accurate. Therefore, a task type's tolerance to MoE layers' routing accuracy varies greatly.

Observation 2. Different task types exhibit markedly different tolerance to inaccurate MoE layers' expert routing. Open-ended tasks (e.g., conversation) require more accurate routing, whereas well-defined tasks (e.g., semantic classification) are substantially more robust.

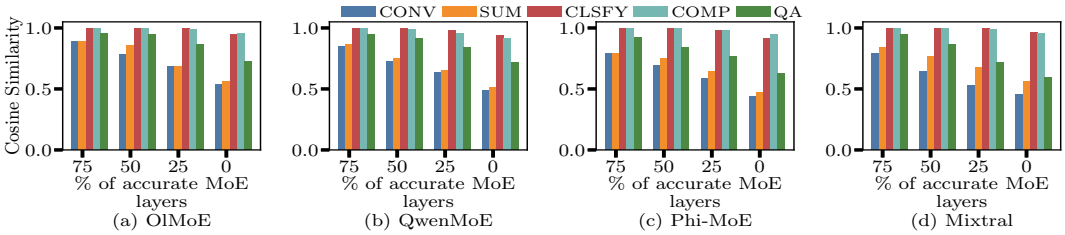


Figure 3: Accuracy with progressive applying of exact token-to-expert routing in MoE layers closer to the output layer.

3 System Design of LiteMoE

Based on the observations in Section 2, we propose LiteMoE, which consists of three components: multi-layer expert prediction, task-aware selective expert loading, and task-aware request scheduling. For each request, a two-tier task identifier, consisting of a keyword fast path and an LLM-based semantic classifier, identifies the task type. The multi-layer expert predictor forecasts the experts required by the next group of MoE layers, while the task-aware selective loader combines these predictions with offline routing-sensitivity profiles and predicted expert usage to determine which experts to place on the GPU. Expert loading is overlapped with ongoing computation, and loaded experts are reused across the predicted layer group. Finally, the task-aware scheduler accounts for expert loading latency when admitting new requests, reducing cross-request interference and SLO violations.

3.1 Multi-layer Expert Prediction

Motivated by Observation 1, we propose a multi-layer prediction mechanism that selects experts for a group of B upcoming layers at once rather than only the next layer. This is facilitated by the fact that MoE gating mechanisms are simple and computationally lightweight, producing outputs with sufficient regularity to allow reliable estimation of future expert selections. Leveraging this property, LiteMoE uses the expert selection from the current layer to estimate the experts likely to be activated in the B upcoming layers. The predicted experts, filtered based on task-specific sensitivity and expert usage frequency (Section 3.2), are collectively migrated to GPU memory in a single transfer operation. These experts remain cached in GPU memory, serving multiple subsequent blocks without additional transfer costs. By amortizing expert migration across several subsequent layers, this approach significantly reduces the number of CPU-to-GPU transfers required during inference. Figure 4 illustrates how LiteMoE leverages the MoE transformer block structure to predict expert selections for multiple future layers in a single step. After the N-th MoE layer processes its tokens and determines the selected experts via the router, this expert selection and the tokens' hidden states, are fed into the predictor. The predictor then generates predicted expert sets for layers $N+1$, $N+2$, ..., up to $N+B$. This prediction step occurs concurrently with the expert computation at block N, ensuring that expert pre-fetching for future layers does not stall ongoing processing.

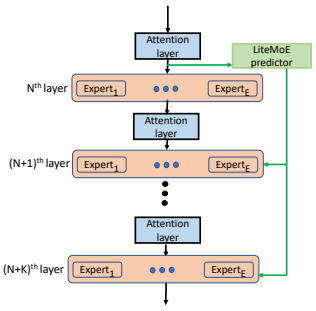


Figure 4: Router predictor.

During inference, when the computation reaches the $(N+1)$-th MoE layer, the conventional router is bypassed. Instead, LiteMoE directly uses the predicted expert set generated earlier as the activation decision for that layer. This process is repeated for each subsequent layer up to $N+B$. As a result, expert loading is minimized to once per B layers, significantly reducing CPU-to-GPU transfer overhead.

Figure 5 illustrates the execution pipelines of Pre-gatedMoE and LiteMoE, emphasizing their expert pre-fetching strategies and impact on CPU-GPU communication. In Pre-gatedMoE, experts are selected during each layer's computation, leading to frequent transfers and increased PCIe pressure. The red dotted line shows the waiting time caused by expert loading, which cannot be overlapped with computation and directly impacts latency. In contrast, LiteMoE predicts experts for the next B layers in advance, pre-fetches them in bulk to GPU memory for reuse.

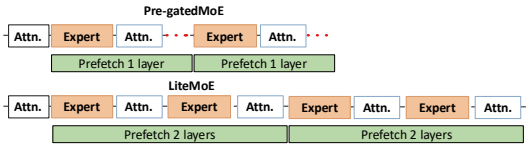


Figure 5: Execution pipeline of pre-gatedMoE and LiteMoE.

However, determining the value of B is a critical design consideration. A small B limits the potential benefits of reducing expert migrations, while an overly large B may degrade routing accuracy, as the predictor's estimation becomes less reliable further into the future. To balance these trade-offs, we conduct an empirical analysis, discussed in detail in Section A.7.1, to evaluate the impact of different B values on routing prediction accuracy, latency, and accuracy. Based on these insights, LiteMoE adopts $B = 4$ as the default configuration, which offers an optimal balance between prediction accuracy and migration reduction.

LiteMoE's router predictor is designed as a lightweight multi-layer perceptron (MLP). To build this predictor, we collect routing decision traces across a variety of workloads with input prompts, capturing the input to each layer and the corresponding output of the gating function. This collected dataset is then used to train the predictor to learn expert selection patterns and accurately forecast future experts. The MLP predictor is trained for each MoE model using model-specific routing traces. During training, we tune hyperparameters such as block size (B), hidden dimension, learning rate, the number of MLP layers, and training iterations. Through this process, we identify the optimal configuration, which consists of 2 MLP layers. The overall training process took around 3 hours. Once trained, the predictor operates with minimal runtime overhead (Figure 10).

For a batch consisting of requests from different task, LiteMoE predicts the top-K experts for each token across B layers. Since these predictions may be inaccurate, we introduce a rollback mechanism. After routing decisions are made, if the required experts are not included in the predicted set, LiteMoE loads the missing experts on demand and re-executes the computation. This rollback mechanism improves accuracy at the cost of additional latency. Our experiments show that rollback is triggered for only 6.6–13.0% of predictions, depending on the model (Table 3), resulting in minimal overall latency overhead.

3.2 Task-aware Selective Expert Loading

From the predictions in Section 3.1, LiteMoE computes the relative frequency of experts by counting how often each expert appears across the predicted layers for a given task type, indicating how likely tokens from that task type are to be routed to each expert. These predictions with relative frequency are then used to filter and load experts based on each task's sensitivity to token-to-expert routing accuracy and expert's usage frequency.

The task-aware expert loader uses task sensitivity profiling to identify the experts required for accuracy-sensitive tasks (details in Section 3.2.1). Only these experts are loaded into GPU memory, minimizing transfers for tasks that are tolerant to routing imprecision. If GPU memory is insufficient to host all experts, experts with low usage frequency are skipped, and their tokens are routed to the next-best available experts (details in Section 3.2.2).

3.2.1 Task-Aware Expert Selection from Predictions

If expert loading latency can be hidden by computation and GPU memory is sufficient to host all predicted experts based on profiled data, LiteMoE loads all predicted experts into GPU memory. Otherwise, LiteMoE activates task-aware expert selection from predictions. Our Observation 2 indicates that certain task types show greater sensitivity to token-to-expert routing accuracy while some task types show little sensitivity to this accuracy. In the context of expert loading for a particular layer, we denote a task as either sensitive or insensitive. LiteMoE has task-aware selective expert loading, which only considers the predicted experts of the sensitive tasks during the expert loading to reduce the expert loading latency while incurring minimal compromise on the accuracy of the model output.

We identify the task type of each incoming request using a lightweight LLM-based semantic classifier (Shah et al., 2025; Atuhurra, 2024), preceded by a keyword fast path. The fast path labels a prompt only when exactly one task template matches it (about 0.05 ms per request); all other prompts, including paraphrased or ambiguous ones, go to the classifier. The classifier operates on

prompt semantics, allowing it to recognize paraphrased and diverse prompts that express the same task without requiring predefined trigger phrases. It outputs both a task label and a confidence score. If the classification confidence is below a predefined threshold or the request does not match any profiled task type, LiteMoE falls back to standard expert loading without task-aware routing relaxation. An identification error therefore harms accuracy only when it assigns a less routing sensitive task type (a *harmful error*); other errors and fallback only reduce efficiency. On an assumed deployment mix, only 1.7% of requests are harmfully misclassified and 9.1% fall back, while throughput and accuracy stay close to those with oracle (ground truth) task labels (Appendix A.5).

LiteMoE's task-aware selective expert loading operates in two phases: (i) an offline profiling phase that characterizes the routing sensitivity of each task type and (ii) an online phase that uses the identified task type and its profiled sensitivity to determine expert loading. We define routing sensitivity at the granularity of MoE layers. During offline profiling, for each task type, we progressively introduce random token-to-expert routing, beginning with the MoE layer closest to the input, following the methodology in §2.3. We compare the outputs produced under accurate and perturbed routing and measure their cosine similarity. If the average similarity remains above a predefined quality threshold (e.g., 0.9), we classify the task type as insensitive to routing perturbations in the affected MoE layers. Otherwise, those layers are treated as routing-sensitive and retain standard expert loading.

Once the offline profiling phase is complete, LiteMoE's task-aware selective expert loading module leverages the task type extraction method to identify the task type associated with each incoming request. It then filters the output of the expert prediction module for each MoE layer and task type based on the determined task task sensitivity. Finally, it selects the remaining experts to be loaded and initiates the loading process.

3.2.2 Usage-Frequency-based Expert Selection

If the memory size is not sufficient to host all of the remaining experts, then LiteMoE activate this usage-frequency-based expert selection. In each iteration of expert loading, LiteMoE first computes the expected number of tokens per expert per task type and assembles the computed data in a 3D array as shown in Figure 6. We obtain the expected number of tokens per expert with an objective to maximize the number of correct token-to-expert routing. We hypothesize that maximizing the number of correctly routed tokens per expert will ensure the least impact on the accuracy.

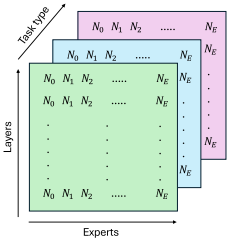


Figure 6: LiteMoE maintains the expected number of tokens per expert for each task type.

Let $N_{i,j}$ be the expected number of tokens that will be routed to expert i in an arbitrary MoE layer for task type j. Also, let $s_{j,\ell} \in \{0, 1\}$ indicate whether tokens of task type j are sensitive to token-to-expert routing accuracy at layer ℓ, let $f_{i,j}$ denote the predicted token-to-expert routing frequency for expert i under task type j, and let T_j be the number of requests of task type j that are currently running. We compute $N_{i,j}$: $N_{i,j} = \left(\sum_{k=1}^{|Q_j^{\text{new}}|} W_{k,j} + T_j \cdot W_{o,j}\right) \cdot s_{j,\ell} \cdot f_{i,j}$, where $W_{k,j}$ denotes the number of input tokens of the k-th request of task type j, Q_j^{new} denote the set of prompts in the batch of task type j, and $W_{o,j}$ denotes the expected number of generated tokens for task type j.

LiteMoE utilizes the equation to determine the expected number of tokens per expert for a given set of requests that are to be scheduled and the requests that are already running by the inference engine. LiteMoE obtains the total expected number of tokens per expert by summing $N_{i,j}$s belonging to all task types. Next, LiteMoE sorts the experts at each MoE layer based on the expected number of tokens likely to be routed in descending order. Subsequently, LiteMoE picks the top L unloaded experts for every layer, where L is determined by the memory budget. It then loads the selected experts into memory.

3.3 Task-aware Request Scheduling

LiteMoE's request scheduler aims to minimize the impact of expert loading on inference latency while protecting the SLOs of running requests. Existing LLM schedulers (e.g., (Kwon et al., 2023; Holmes et al., 2024; Agrawal et al., 2024; Zhong et al., 2024)) do not explicitly account for expert loading. In memory-efficient MoE serving, however, admitting a new request can trigger expert

transfers and introduce additional computation and all-to-all communication, increasing the latency of concurrently running requests. The impact varies across tasks because task types differ in generation length and in the amount of expert loading required by task-aware selective loading. Therefore, LiteMoE jointly considers user-imposed latency SLOs, task-specific profiled generation lengths, the new request's input length, and effective expert loading latency when making admission decisions. When a waiting request has a relaxed SLO and running requests are expected to finish soon, delaying its admission can avoid unnecessary cross-request interference and reduce SLO violations.

Let W be the number of input tokens in a new request to be scheduled, G_i be the expected number of remaining tokens to be generated by the ith request, ΔE be the expert loading latency introduced by scheduling a new request, and c be the average incremental computation and communication latency per token under batched execution. We use task-specific profiling to obtain G_i. When a request is scheduled, G_i is initialized to the average number of tokens generated by requests of the corresponding task type. After each iteration of token generation, G_i is decremented. If G_i reaches 0 while the request is still running, G_i is reset to a value proportional (e.g., 10%) to its initial value. We also use profiled values for ΔE and c.

Since expert loading can overlap with computation, only the non-overlapped portion of the loading latency contributes to the end-to-end latency. Let ΔE_{eff} denote the effective expert loading latency after overlap: $\Delta E_{\text{eff}} = \max(0, \Delta E - T_{\text{hide}})$, where T_{hide} denotes the amount of computation time available to hide expert loading latency. Also, let n_i denote the number of existing running requests whose generation completes after the ith request, i.e., requests that will still generate at least G_i tokens before the completion of the ith request. We express the expected latency of the ith request as follows: $t_i = r_i + \Delta E_{\text{eff}} + (W + n_i \cdot G_i) \cdot c$, where r_i denotes the runtime of request i so far. For a newly scheduled request, r_i is initialized to zero and incremented during execution.

At each iteration, LiteMoE picks the request with the most stringent time-to-first-token (TTFT) latency SLO and attempts to schedule the request. It first checks if scheduling the request will exceed the total number of tokens that the inference engine can process. If yes, the request is kept in the waiting queue. Otherwise, the request is scheduled if it is expected to meet its SLO and it will not lead to SLO violations of the requests already scheduled.

4 Performance Evaluation

Unless otherwise stated, the experiment settings are the same as those in Section 2. We also tested the DeepSeek-V2-Lite model. We used 1 and 2 GPUs for OlMoE and QwenMoE, respectively. For DeepSeek, Phi-MoE and Mixtral models, we used 4 GPUs. In the multi-GPU setting, we used tensor parallelism. The K values of top-K expert routing in Mixtral, OlMoE, DeepSeek, Phi-MoE, and Qwen-MoE are 2, 8, 6, 2, and 4, respectively. We also evaluate on an AWS g6.12xlarge instance with four 24 GB NVIDIA L4 GPUs (Appendix A.3), where expert offloading is required (Appendix A.4). We evaluated LiteMoE against Pre-gated MoE (Hwang et al., 2024), MoEOnDemand (acc, 2022), and MoEPrefetch (Shen et al., 2023). LiteMoE and all baseline methods were implemented on top of the vLLM (Kwon et al., 2023) framework. Pre-gatedMoE uses its pretrained gate to predict expert selection for the next layer, which is then utilized to pre-load the necessary experts. MoEOnDemand loads the experts as determined by the respective MoE layer gate. MoEPrefetch proactively pre-loads all experts to the GPU while the current layer is executing, without waiting for the gate decision. If the memory is not sufficient to host all experts, then a token uses the best local expert. We utilized the same datasets and MoE models as outlined in §2.1. B was empirically set to 4. We set the latency SLO of each request to twice of its average inference latency, following the SLO setting methodology in AlpaServe (Li et al., 2023b). The request rate of DeepSeek was set to 0.6 req/s.

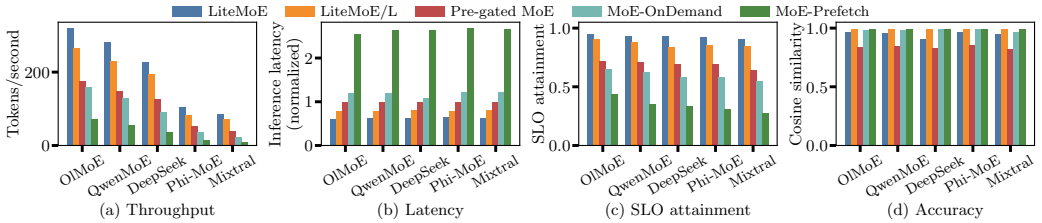


Figure 7: End-to-end inference performance on A100.

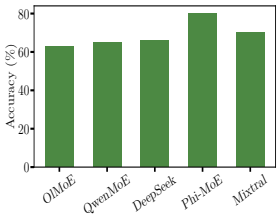


Figure 8: LiteMoE router prediction accuracy.

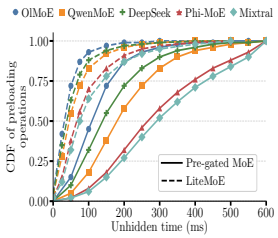


Figure 9: CDF of preloading operations.

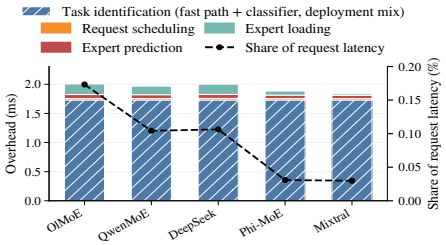


Figure 10: LiteMoE decision-making overhead on A100.

End-to-End Performance. Figure 7(b) shows the end-to-end inference latency of the methods across the five models in A100. LiteMoE consistently achieves the lowest latency, reducing inference time by 40–54% compared to Pre-gatedMoE. The improvements over MoEOnDemand are even more significant, with LiteMoE achieving 48–63% lower latency across models. Against MoEPrefetch, which preloads all experts before computation and is the slowest method, LiteMoE reduces latency by 76 to 79%. The task-aware selective expert loading method reduces the expert loading time, and hence the latency.

Figure 7(c) shows the SLO attainment. LiteMoE improves the SLO attainment over Pre-gatedMoE and MoEOnDemand by 24 and 33 percentage points on average, respectively, across the five models, and achieves $\sim$2.7$\times$ higher SLO attainment than MoEPrefetch, due to the same reasons explained above. Figure 7(d) shows the average inference accuracy achieved by different methods. MoEOnDemand and MoEPrefetch load all required experts and therefore achieve 100% accuracy. LiteMoE employs task-aware selective expert loading and skips loading experts for accuracy-insensitive tasks, resulting in a slight accuracy degradation (by 3–9 percentage points) while achieving higher efficiency gains. In contrast, Pre-gatedMoE may incorrectly predict the required experts, leading to an accuracy degradation of 14–18 percentage points. Across task types, LiteMoE maintains output similarity near 0.9 while consistently outperforming Pre-gatedMoE in quality. Detailed per-task accuracy results are reported in Appendix A. Against gold references, summarization ROUGE-L drops by 1.0 to 1.9 points with LiteMoE versus 2.5 to 4.0 with Pre-gated MoE on A100 (Appendix A.2).

In the figures, we also include the performance of LiteMoE/L, which represents LiteMoE without the task-aware selective expert loading subsystem, and incurs no accuracy loss. Compared with LiteMoE, LiteMoE/L reduces throughput by 15–21%, increases latency by 21–28%, and decreases SLO attainment by 4–7 percentage points. Compared to Pre-gated MoE, it still improves throughput by 1.51–1.85$\times$, reduces inference latency by 23–33%, improves SLO attainment by 15–21 percentage points and accuracy by 13–17 percentage points across all models.

Figure 8 evaluates router prediction accuracy by comparing the predicted experts with the experts selected by the router and averaging accuracy across the four predicted layers. Mixtral and Phi-MoE achieve approximately 70% and 80% accuracy, respectively, while OlMoE, QwenMoE, and DeepSeek achieve around 63%, 65%, and 66%. Figure 9 shows the CDF of the communication time that remains unhidden after computation overlap. LiteMoE places a substantially larger fraction of preloading operations in the low-unhidden-time region than Pre-gatedMoE; for example, 94% of LiteMoE operations versus 55% of Pre-gatedMoE operations have no more than 200 ms of unhidden time, demonstrating more effective communication–computation overlap. Figure 10 reports LiteMoE's decision-making overhead. Task identification dominates at 1.73 ms per request; the other components add 0.11 to 0.28 ms, so the total overhead is only 0.030 to 0.173% of request latency. The expert loading component is larger for OlMoE, QwenMoE, and DeepSeek because they have more experts per layer.

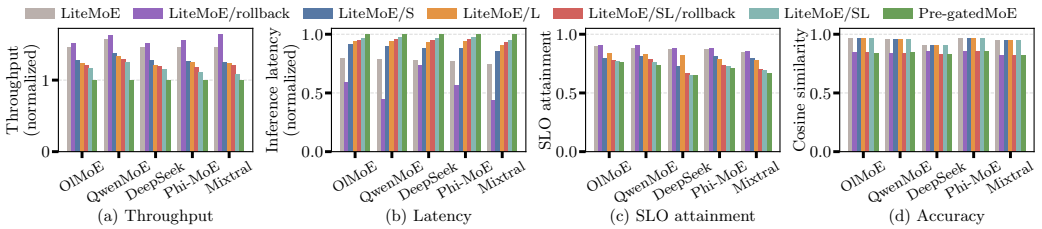


Figure 11: Ablation study.

Ablation Study. In Figure 11, LiteMoE/S, LiteMoE/L, and LiteMoE/SL denote LiteMoE without the scheduler, selective expert loading, and both, respectively; /rollback denotes LiteMoE without rollback. Multi-layer expert prediction provides the largest performance gain, while scheduling and selective loading provide comparable additional benefits. The scheduler improves throughput by 14–16%, reduces latency by 11–14%, and increases SLO attainment by 5–10 percentage points. Selective loading improves throughput by 17–21%, reduces latency by 15–19%, and increases SLO attainment by 5–8 percentage points with negligible accuracy loss. Compared with Pre-gatedMoE, multi-layer prediction alone (LiteMoE/SL) improves throughput by 8–24%, reduces latency by 7–13%, and increases SLO attainment by 4–5 percentage points, with an 8–13% accuracy increase. While rollback preserves model accuracy, it introduces modest performance overhead. Compared with LiteMoE/SL/rollback, LiteMoE/SL increases latency by 1.9%, reduces throughput by 5% and SLO attainment by 1.6 percentage points, while improving accuracy by 11 percentage points on average. Similarly, rollback increases LiteMoE's latency by 16%, reduces throughput by 5% and SLO attainment by 2.4 percentage points, while improving accuracy by 6.8 percentage points. Thus, rollback provides a configurable accuracy–performance trade-off. The scheduler provides greater benefits as expert size increases, with Mixtral, which has the largest experts, achieving the largest performance gain.

5 Related Work

Prior work has developed increasingly scalable MoE architectures to expand model capacity while limiting computation (Shazeer et al., 2017; Zuo et al., 2022; Li et al., 2020; Nie et al., 2022; Qin et al., 2020; He et al., 2023; Dai et al., 2022; Chowdhury et al., 2023; Zhou et al., 2023; Lewis et al., 2021; Gao et al., 2022; Wang et al., 2022; Dai et al., 2024; Kudugunta et al., 2021; Kang et al., 2025; Zhao et al., 2024; Zhang et al., 2022; Wang et al., 2024; 2025; Puigcerver et al., 2024; Wu et al., 2025). Prior systems improve MoE inference through parallelism, communication optimization, expert offloading, caching, and prefetching (Shen et al., 2023; Rajbhandari et al., 2022; Hwang et al., 2023; Li et al., 2023a;c; Zhong et al., 2024; Agrawal et al., 2024; Eliseev & Mazur, 2023; Hwang et al., 2024; Xue et al., 2024; Yi et al., 2023; Cui et al., 2023; Fang et al., 2025; Cao et al., 2025; He et al., 2026; Kong et al., 2024; acc, 2022; Dai et al., 2024; Du et al., 2024; Li et al., 2024). DeepSpeed-MoE (Rajbhandari et al., 2022) uses tensor and expert slicing for distributed execution, while DeepSeekMoE (Dai et al., 2024) overlaps communication and computation. Tutel (Hwang et al., 2023) dynamically adapts parallelism and pipelining to MoE workloads. Memory-efficient systems overlap expert transfers from CPU to GPU with computation (Eliseev & Mazur, 2023; Hwang et al., 2024; Xue et al., 2024; Yi et al., 2023; Zhu et al., 2025; Cao et al., 2025; Fang et al., 2025). SwapMoE (Kong et al., 2024) maintains dynamic virtual experts under configurable memory budgets, while prediction-driven approaches reduce loading latency through expert prefetching or caching. Yu *et al.* (Yu et al., 2026) further propose fine-grained expert offloading for MoE-based LLM serving. Compared to these methods, LiteMoE employs task-aware, multi-layer expert prediction and loading, focusing on improving system throughput and reducing latency in resource-constrained environments.

6 Conclusion

We presented LiteMoE, a memory-efficient MoE inference system motivated by two empirical findings: expert routing across nearby layers is predictable, and tasks differ in their sensitivity to routing approximation. LiteMoE predicts experts for groups of future layers, coalescing expert transfers and increasing the opportunity to overlap communication with computation. Under resource constraints, it further uses task-aware routing-sensitivity profiles and predicted expert usage to selectively load the experts that matter most for output quality. Additionally, we introduce task-aware request scheduling that accounts for expert loading latency to minimize its impact on existing requests' SLO attainment. Our evaluation shows that these mechanisms substantially improve throughput and latency over prior memory-efficient MoE inference approaches while providing an explicit quality–efficiency tradeoff. More broadly, the results suggest that routing structure can be treated as an inference-time optimization signal for making large MoE models practical on memory-constrained GPU platforms.

AI Assistance Disclosure

Generative AI tools were used to improve the clarity, grammar, and presentation of portions of the manuscript. The authors reviewed and revised all AI-assisted text and take full responsibility for the final content.

References

HuggingFace accelerate. https://huggingface.co/docs/accelerate, 2022.

Asma Ben Abacha. Medquad: Medical question answering dataset. https://github.com/abachaa/MedQuAD, 2019. Accessed: 2026-06-09.

Marah Abdin, Jyoti Aneja, Hany Awadalla, Ahmed Awadallah, Ammar Ahmad Awan, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Jianmin Bao, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with sarathi-serve. In *Proc. of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 117–134, 2024.

Jesse Atuhurra. Zero-shot retrieval of user intent in human-robot interaction with large language models. In *Proceedings of the IEEE International Conference on Multimedia Information Processing and Retrieval (MIPR)*, 2024.

Warda Bilal. Smart university query classification dataset. https://www.kaggle.com/datasets/wardabilal/smart-university-query-classification-dataset, 2026. Accessed: 2026-06-09.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E. Gonzalez, Matei Zaharia, and Ion Stoica. 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*, ASPLOS '25, pp. 715–730, New York, NY, USA, March 2025. Association for Computing Machinery. ISBN 979-8-4007-0698-1. doi: 10.1145/3669940.3707267.

Daniel Cer et al. Semantic textual similarity benchmark. https://huggingface.co/datasets/sentence-transformers/stsb. Accessed: 2026-06-09.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Mohammed Nowaz Rabbani Chowdhury, Shuai Zhang, Meng Wang, Sijia Liu, and Pin-Yu Chen. Patch-level routing in mixture-of-experts is provably sample-efficient for convolutional neural networks. In *Proc. of International Conference on Machine Learning*, pp. 6074–6114. PMLR, 2023.

Weihao Cui, Zhenhua Han, Lingji Ouyang, Yichuan Wang, Ningxin Zheng, Lingxiao Ma, Yuqing Yang, Fan Yang, Jilong Xue, Lili Qiu, et al. Optimizing dynamic neural networks with brainstorm. In *Proc. of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 797–815, 2023.

Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022.

Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, volume 1, 2024.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 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)*, pp. 4171–4186, 2019.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 293–302, 2019.

Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 33:18518–18529, 2020.

Zhixu Du, Shiyu Li, Yuhao Wu, Xiangyu Jiang, Jingwei Sun, Qilin Zheng, Yongkai Wu, Ang Li, Hai Li, and Yiran Chen. Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models. In *Proceedings of Machine Learning and Systems*, volume 6, pp. 224–238, 2024.

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

Zhiyuan Fang, Yuegui Huang, Zicong Hong, Yufeng Lyu, Wuhui Chen, Yue Yu, Fan Yu, and Zibin Zheng. Klotski: Efficient Mixture-of-Expert Inference via Expert-Aware Multi-Batch Pipeline. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ASPLOS '25, pp. 574–588, New York, NY, USA, March 2025. Association for Computing Machinery. ISBN 979-8-4007-1079-7. doi: 10.1145/3676641.3716261.

Ze-Feng Gao, Peiyu Liu, Wayne Xin Zhao, Zhong-Yi Lu, and Ji-Rong Wen. Parameter-efficient mixture-of-experts architecture for pre-trained language models. In *Proceedings of the 29th International Conference on Computational Linguistics (COLING 2022)*, 2022.

Shwai He, Run-Ze Fan, Liang Ding, Li Shen, Tianyi Zhou, and Dacheng Tao. Merging experts into one: Improving computational efficiency of mixture of experts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, 2023.

Xin He, Shunkang Zhang, Kaijie Tang, Shaohuai Shi, Yuxin Wang, Zihao Zeng, Zhenheng Tang, Xiaowen Chu, Haiyan Yin, Ivor Tsang, and Ong Yew Soon. ExpertFlow: Efficient Mixture-of-Experts Inference via Predictive Expert Caching and Token Scheduling. In *Proceedings of the 63rd ACM/IEEE Design Automation Conference*, 2026. To appear.

Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

Haiyang Huang, Newsha Ardalani, Anna Sun, Liu Ke, Hsien-Hsin S. Lee, Shruti Bhosale, Carole-Jean Wu, and Benjamin Lee. Toward efficient inference for mixture of experts. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, volume 37 of *NIPS '24*, pp. 84033–84059, Red Hook, NY, USA, December 2024. Curran Associates Inc. ISBN 979-8-3313-1438-5.

Changho Hwang, Wei Cui, Yifan Xiong, Ziyue Yang, Ze Liu, Han Hu, Zilong Wang, Rafael Salas, Jithin Jose, Prabhat Ram, HoYuen Chau, Peng Cheng, Fan Yang, Mao Yang, and Yongqiang Xiong. Tutel: Adaptive mixture-of-experts at scale. In D. Song, M. Carbin, and T. Chen (eds.), *Proceedings of Machine Learning and Systems*, volume 5, pp. 269–287. Curan, 2023.

Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, Ting Cao, and Mao Yang. Pre-Gated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference. In *Proceedings of the 51st Annual International Symposium on Computer Architecture*, ISCA '24, pp. 1018–1031, Buenos Aires, Argentina, 2024. IEEE Press. ISBN 979-8-3503-2658-1. doi: 10.1109/ISCA59077.2024.00078.

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

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, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts. *arXiv preprint arXiv: 2401.04088*, 2024.

Junmo Kang, Leonid Karlinsky, Hongyin Luo, Zhen Wang, Jacob Hansen, James Glass, David Cox, Rameswar Panda, Rogerio Feris, and Alan Ritter. Self-MoE: Towards compositional large language models with self-specialized experts. In *Proc. of ICLR*, 2025.

Young Jin Kim, Ammar Ahmad Awan, Alexandre Muzio, Andres Felipe Cruz Salinas, Liyang Lu, Amr Hendy, Samyam Rajbhandari, Yuxiong He, and Hany Hassan Awadalla. Scalable and efficient moe training for multitask multilingual models. *arXiv preprint arXiv:2109.10465*, 2021.

Rui Kong, Yuanchun Li, Qingtian Feng, Weijun Wang, Xiaozhou Ye, Ye Ouyang, Linghe Kong, and Yunxin Liu. SwapMoE: Serving off-the-shelf MoE-based large language models with tunable memory budget. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, August 2024.

Sneha Kudugunta, Yanping Huang, Ankur Bapna, Maxim Krikun, Dmitry Lepikhin, Minh-Thang Luong, and Orhan Firat. Beyond distillation: Task-level mixture-of-experts for efficient inference. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 3577–3599, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.304.

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

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.829.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling giant models with conditional computation and automatic sharding. In *International Conference on Learning Representations*, 2021.

Mike Lewis, Shruti Bhosale, Tim Dettmers, Naman Goyal, and Luke Zettlemoyer. Base layers: Simplifying training of large, sparse models. In *Proc. of International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Dingcheng Li, Xu Li, Jun Wang, and Ping Li. Video recommendation with multi-gate mixture of experts soft actor critic. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1553–1556, 2020.

Jiamin Li, Yimin Jiang, Yibo Zhu, Cong Wang, and Hong Xu. Accelerating Distributed MoE Training and Inference with Lina. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pp. 945–959, 2023a. ISBN 978-1-939133-35-9.

Pingzhi Li, Zhenyu Zhang, Prateek Yadav, Yi-Lin Sung, Yu Cheng, Mohit Bansal, and Tianlong Chen. Merge, then compress: Demystify efficient smoe with hints from its routing policy. In *Proc. of ICLR*, 2024.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Z. Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Alpaserve: Statistical multiplexing with model parallelism for deep learning serving. In *USENIX Symposium on Operating Systems Design and Implementation*, 2023b.

Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *Proc. of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp. 663–679, Boston, MA, July 2023c. USENIX Association. ISBN 978-1-939133-34-2.

Shiyi Liu, Haiying Shen, Shuai Che, Mahdi Ghandi, and Mingqin Li. AIMS: Cost-efficient LLM-based agent deployment in hybrid cloud-edge environments. In *Proceedings of the 21st European Conference on Computer Systems (EuroSys)*, 2026.

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Morrison, Sewon Min, Weijia Shi, Evan Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, Ali Farhadi, Noah A. Smith, Pang Wei Koh, Amanpreet Singh, and Hannaneh Hajishirzi. OLMoE: Open mixture-of-experts language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Xiaonan Nie, Pinxue Zhao, Xupeng Miao, Tong Zhao, and Bin Cui. HetuMoE: An efficient trillion-scale mixture-of-expert distributed training system. *arXiv preprint arXiv:2203.14685*, 2022.

Alexander Nikitin, Jannik Kossen, Yarin Gal, and Pekka Marttinen. Kernel language entropy: Fine-grained uncertainty quantification for llms from semantic similarities. In *Advances in Neural Information Processing Systems*, volume 37 of *NIPS '24*, pp. 8901–8929. Curran Associates Inc., 2024.

OpenAssistant. Openassistant conversations dataset. `https://huggingface.co/datasets/OpenAssistant/oasst1`, 2023. Accessed: 2026-06-09.

Joan Puigcerver, Carlos Riquelme, Basil Mustafa, and Neil Houlsby. From sparse to soft mixtures of experts. In *Proc. of ICLR*, 2024.

Zhen Qin, Yicheng Cheng, Zhe Zhao, Zhe Chen, Donald Metzler, and Jingzheng Qin. Multitask mixture of sequential experts for user activity streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 3083–3091, 2020.

Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. In *Proceedings of the 39th International Conference on Machine Learning*, pp. 18332–18346. PMLR, June 2022.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2383–2392. Association for Computational Linguistics, 2016.

Chirag Shah, Ryen White, Reid Andersen, Georg Buscher, Scott Counts, Sarkar Das, Ali Montazer, Sathish Manivannan, Jennifer Neville, Nagu Rangan, et al. Using large language models to generate, validate, and apply user intent taxonomies. *ACM Transactions on the Web*, 19(3):1–29, 2025.

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

Liang Shen, Zhihua Wu, WeiBao Gong, Hongxiang Hao, Yangfan Bai, HuaChao Wu, Xinxuan Wu, Jiang Bian, Haoyi Xiong, Dianhai Yu, and Yanjun Ma. SE-MoE: A scalable and efficient mixture-of-experts distributed training and inference system. *arXiv preprint arXiv 2205.10034*, 2023.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 8815–8821, 2020.

Emily Silcock, Abhishek Arora, and Melissa Dell. A massive scale semantic similarity dataset of historical english. In *Advances in Neural Information Processing Systems*, volume 36, pp. 58113–58127, 2024.

Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Proc. of NeurIPS*, 2020.

Qwen Team. Qwen1.5-MoE: Matching 7B model performance with 1/3 activated parameters", February 2024.

David Wallach. Financial tweets sentiment. `https://www.kaggle.com/datasets/davidwallach/financial-tweets`, 2018. Accessed: 2026-06-09.

An Wang, Xingwu Sun, Ruobing Xie, Shuaipeng Li, Jiaqi Zhu, Zhen Yang, Pinxue Zhao, JN Han, Zhanhui Kang, Di Wang, Naoaki Okazaki, and Cheng-zhong Xu. HMoE: Heterogeneous mixture of experts for language modeling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2025.

Kuan-Chieh Wang, Daniil Ostashev, Yuwei Fang, Sergey Tulyakov, and Kfir Aberman. Moa: Mixture-of-attention for subject-context disentanglement in personalized image generation. In *Proc. of SIGGRAPH Asia 2024 Conference Papers*, pp. 1–12, 2024.

Yaqing Wang, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, 2022.

Chenpeng Wu, Qiqi Gu, Heng Shi, Jianguo Yao, and Haibing Guan. Samoyeds: Accelerating MoE Models with Structured Sparsity Leveraging Sparse Tensor Cores. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 293–310, New York, NY, USA, March 2025. Association for Computing Machinery. ISBN 979-8-4007-1196-1. doi: 10.1145/3689031.3717455.

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

Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu. EdgeMoE: Empowering sparse large language models on mobile devices. *IEEE Transactions on Mobile Computing*, 24:7059–7073, 2023.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *Proc. of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, 2022.

Hanfei Yu, Xingqi Cui, Hong Zhang, Hao Wang, and Hao Wang. Taming Latency-Memory Trade-Off in MoE-Based LLM Serving via Fine-Grained Expert Offloading. In *Proceedings of the 21st European Conference on Computer Systems*, EUROSYS '26, pp. 176–191, New York, NY, USA, April 2026. Association for Computing Machinery. ISBN 979-8-4007-2212-7. doi: 10.1145/3767295.3769319.

Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter Liu. Pegasus: Pre-training with extracted gap-sentences for abstractive summarization. In *Proc. of the International Conference on Machine Learning*, pp. 11328–11339. PMLR, 2020.

Minjia Zhang, Conglong Li, Xiaoxia Wu, Zhewei Yao, and Yuxiong He. SaMoE: Parameter efficient MoE language models via self-adaptive expert combination. In *Proceedings of ICLR*, 2022.

Hao Zhao, Zihan Qiu, Huijia Wu, Zili Wang, Zhaofeng He, and Jie Fu. HyperMoE: Towards better mixture of experts via transferring among experts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Proc. of NeurIPS*, volume 36, pp. 46595–46623, 2023.

Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 193–210, 2024.

Yanqi Zhou, Nan Du, Yanping Huang, Daiyi Peng, Chang Lan, Da Huang, Siamak Shakeri, David So, Andrew M. Dai, Yifeng Lu, Zhifeng Chen, Quoc V Le, Claire Cui, James Laudon, and Jeff Dean. Brainformers: Trading simplicity for efficiency. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 42531–42542. PMLR, July 2023.

Ruidong Zhu, Ziheng Jiang, Chao Jin, Peng Wu, Cesar A. Stuardo, Dongyang Wang, Xinlei Zhang, Huaping Zhou, Haoran Wei, Yang Cheng, Jianzhe Xiao, Xinyi Zhang, Lingjun Liu, Haibin Lin, Li-Wen Chang, Jianxi Ye, Xiao Yu, Xuanzhe Liu, Xin Jin, and Xin Liu. MegaScale-Infer: Efficient Mixture-of-Experts Model Serving with Disaggregated Expert Parallelism. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, pp. 592–608, New York, NY, USA, August 2025. Association for Computing Machinery. ISBN 979-8-4007-1524-2. doi: 10.1145/3718958.3750506.

Simiao Zuo, Xiaodong Liu, Jian Jiao, Young Jin Kim, Hany Hassan, Ruofei Zhang, Tuo Zhao, and Jianfeng Gao. Taming sparsely activated transformer with stochastic experts. In *Proc. of ICLR*, 2022.

A Additional Experiments

Quality metrics. Besides the cosine-similarity accuracy used in the main text, we report two task-level views. (i) Agreement with the full-model output: label agreement for CLSFY and COMP, exact match (EM) and F1 for QA, and win-plus-tie rate for the open-ended tasks SUM and CONV, judged by a BERT-based model. (ii) Quality against gold references: accuracy for CLSFY, EM and F1 for QA, Spearman correlation and accuracy for COMP, MT-Bench score for CONV (judged by GPT-4, following the standard MT-Bench protocol), ROUGE-L for SUM, and perplexity increase relative to the full model. "Full model" denotes exact routing with all experts available.

Evaluation data. Due to the computational cost of the additional experiments, the task-level quality (Appendix A.2), expert offloading (Appendix A.4), task identification (Appendix A.5), incremental ablation (Appendix A.6), and prediction block size (Appendix A.7.1) experiments were conducted on a randomly sampled subset containing 50% of the evaluation data described in Section 2.1. All systems are evaluated on the same subset, so the comparisons among them remain consistent.

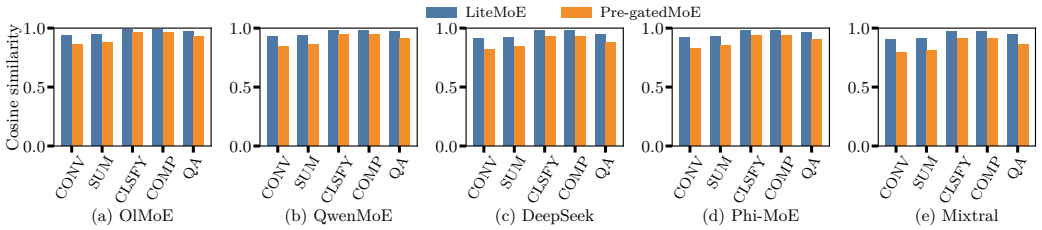


Figure 12: Accuracy for different task types on A100.

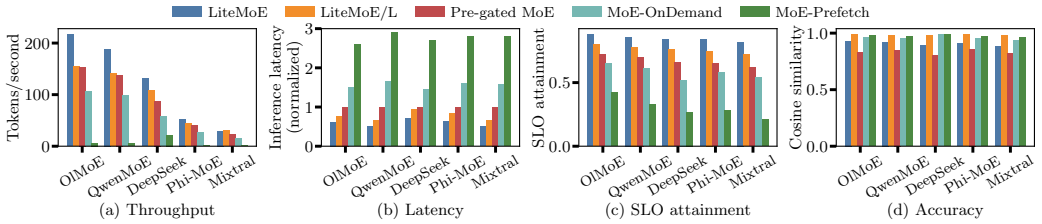


Figure 13: End-to-end inference performance on L4.

A.1 Inference Accuracy

Figure 12 show the cosine similarity of each task type. The cosine similarity metric reflects how closely the partially loaded output approximates the fully loaded output. As shown in the figure, LiteMoE consistently achieves similarity scores near 0.9 across all tasks, indicating that the generated outputs closely match the reference outputs. For classification and comparison tasks, LiteMoE attains near-perfect accuracy. For conversation and summarization tasks, the average similarity across all models is approximately 91%. Across all task types, LiteMoE consistently achieves 3–11 higher accuracy percentage points than Pre-gatedMoE. The lowest accuracy for these tasks is observed in the Mixtral model due to its top-k (k=2) routing strategy and relatively small expert pool (8 experts).

A.2 Task-Level Quality

Figure 15 reports task-level agreement with full-model outputs, and Figure 16 and Table 1 report task quality against gold references. On both GPUs and for every task, LiteMoE stays closer to the full model than Pre-gated MoE. For summarization, LiteMoE loses 1.0–1.9 ROUGE-L points on A100 and 0.8–2.3 points on L4, compared with 2.5–4.0 and 2.8–5.3 points for Pre-gated MoE.

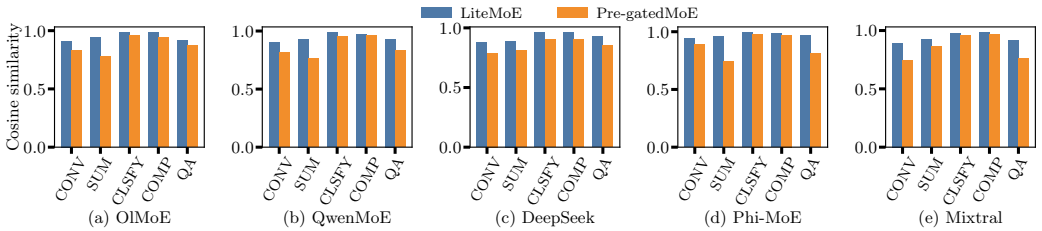
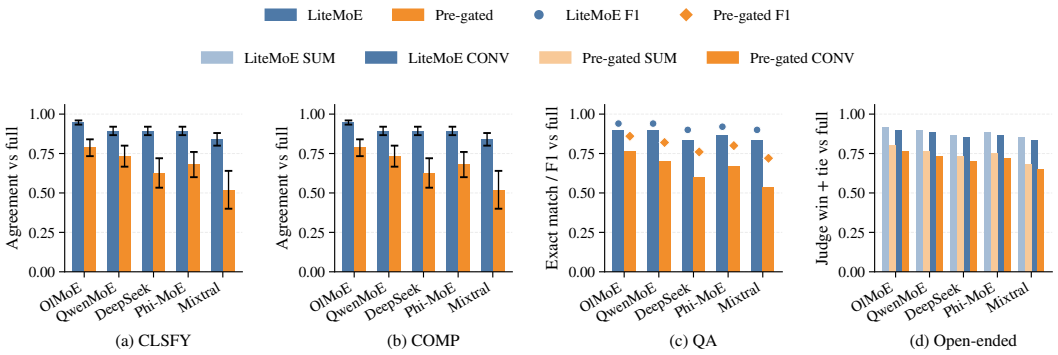
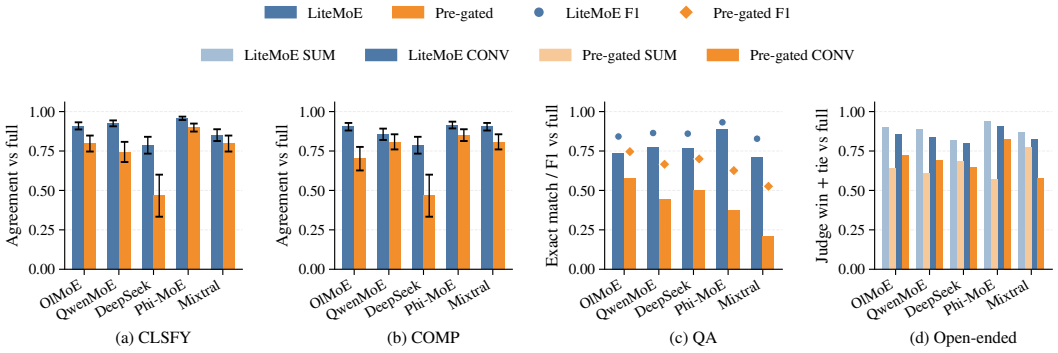


Figure 14: Accuracy for different task types on L4.



(i) A100



(ii) L4

Figure 15: Task-level agreement with full-model outputs on A100 (i) and L4 (ii): (a) CLSFY and (b) COMP label agreement, (c) QA exact match (bars) and F1 (markers), and (d) BERT-judged win-plus-tie rate for the open-ended tasks SUM and CONV.

A.3 Performance Resource-Constrained GPUs

For these experiments, we use an AWS g6.12xlarge instance equipped with an AMD EPYC CPU, 192 GB of host memory, and four NVIDIA L4 GPUs with 24 GB each, running Ubuntu 22.04. Figures 13 and 14 show the performance on L4 GPUs. We observe trends similar to those on A100 GPUs; however, due to the smaller memory capacity of L4, both efficiency and accuracy degrade compared to A100. Specifically, for LiteMoE, the throughput decreases by 24–48%, latency increases by 8–41%, SLO attainment decreases by 2–3 percentage points, and accuracy decreases by 2–7 percentage points across five models.

Despite the more constrained memory environment, LiteMoE consistently outperforms Pre-gated MoE and MoEOnDemand by 28–48%, 91–126%, respectively across all models, and achieves 6–34× higher throughput than MoEPrefetch, with larger gains than those observed on A100 GPUs. These results demonstrate that LiteMoE achieves greater benefits in memory-constrained environments.

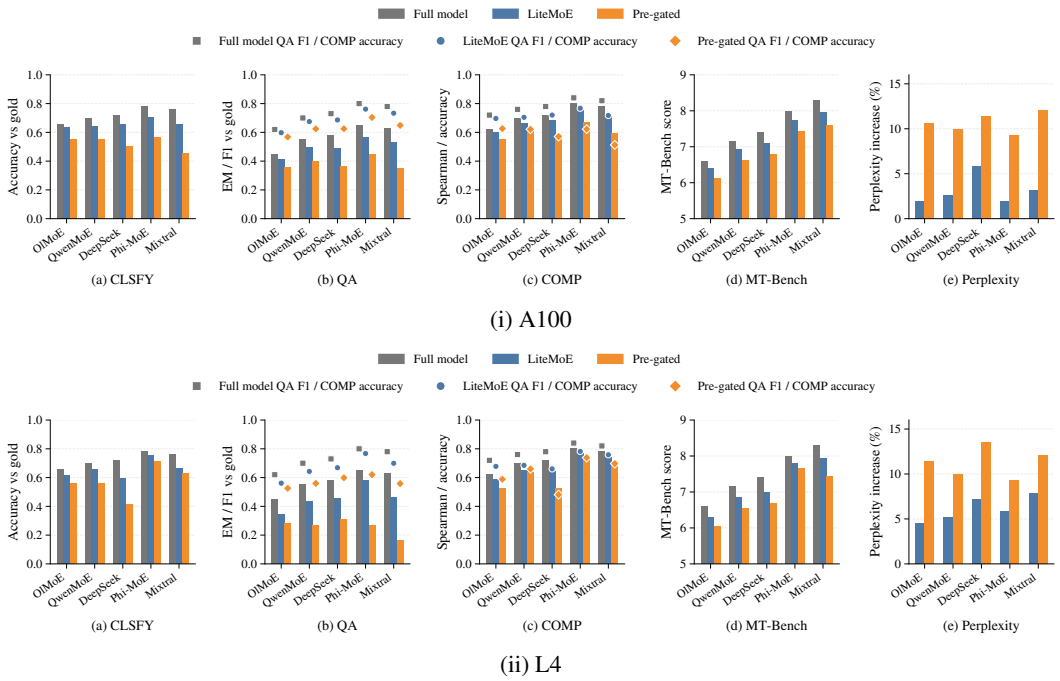


Figure 16: Task-level quality against gold references on A100 (i) and L4 (ii). Bars compare the full model, LiteMoE, and Pre-gated MoE. QA bars show exact match and markers show F1; COMP bars show Spearman correlation and markers show accuracy. CONV quality is the MT-Bench score judged by GPT-4. SUM results are reported in Table 1.

Table 1: SUM ROUGE-L against gold references. The full model scores 0.250 for every model on both GPUs; parentheses give the decrease relative to the full model in points.

	A100		L4	
Model	LiteMoE	Pre-gated MoE	LiteMoE	Pre-gated MoE
OlMoE	**0.240** (1.0)	0.225 (2.5)	**0.238** (1.2)	0.205 (4.5)
QwenMoE	**0.238** (1.2)	0.221 (2.9)	**0.236** (1.4)	0.201 (4.9)
DeepSeek	**0.233** (1.7)	0.217 (3.3)	**0.227** (2.3)	0.210 (4.0)
Phi-MoE	**0.235** (1.5)	0.219 (3.1)	**0.242** (0.8)	0.197 (5.3)
Mixtral	**0.231** (1.9)	0.210 (4.0)	**0.234** (1.6)	0.222 (2.8)

A.4 Necessity of Expert Offloading

Figure 17 quantifies when offloading is required. Without offloading, the remaining KV-cache capacity is small on L4 (e.g., 32K tokens for Phi-MoE), and four L4 GPUs cannot host Mixtral at all. Offloading allows each model to run within the GPU counts used in our evaluation, at a latency cost relative to a deployment in which all experts are resident.

A.5 Task Identification

Classification performance. Table 2 and Figure 18 evaluate task identification on in-distribution, cross-dataset, paraphrased (task keywords removed), mixed-task, and out-of-distribution requests. The deployment mix approximates realistic traffic by weighting in-distribution, paraphrased, mixed-task, and out-of-distribution requests by 70%, 20%, 5%, and 5%, respectively; cross-dataset requests serve as a stress test and are excluded from the mix. Following Section 2.3, routing sensitivity orders the tasks as CONV, SUM $>$ QA $>$ CLSFY, COMP; a harmful error moves a request down this order and is the only error that relaxes routing incorrectly. On in-distribution requests, 98.9% of classified requests are labeled correctly and only 0.3% of all requests are harmfully misclassified.

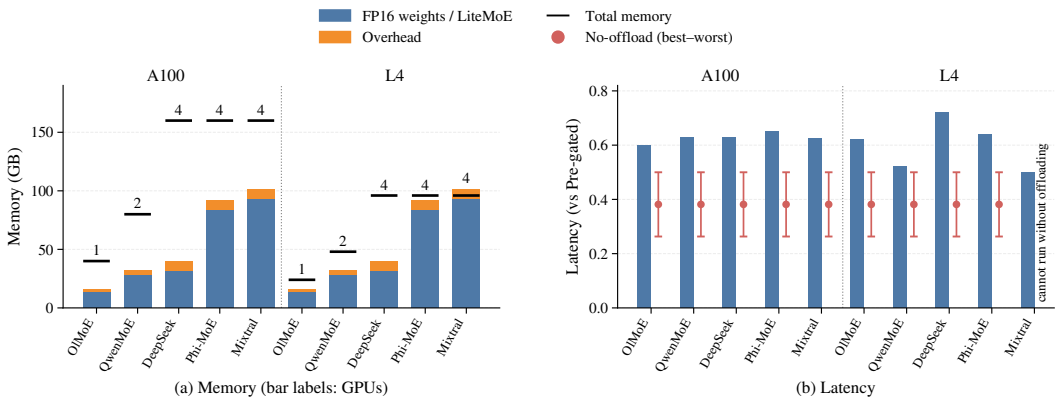


Figure 17: Memory requirements and latency with and without expert offloading. In the order OlMoE, QwenMoE, DeepSeek, Phi-MoE, and Mixtral, KV capacities without offloading are (184, 241, 3878, 520, 447) K tokens on A100 and (62, 78, 1820, 32, 0) K tokens on L4 for the configurations shown. The maximum GPU counts for which offloading is required are (none, none, none, 2, 2) on A100 and (none, 1, 1, 3, 4) on L4, respectively. The four-L4 Mixtral configuration cannot run without offloading.

Table 2: Task-identification performance across evaluation sets. Fallback is the share of requests routed to standard expert loading; accuracy and macro-F1 are computed over classified (non-fallback) requests. A harmful error assigns a request a task type with lower routing sensitivity than its true type. Fast-path share is the fraction of requests labeled without invoking the classifier. Latency columns report per-request identification time on each device.

Evaluation set	Fallback (%)	Accuracy (classified, %)	Macro-F1 (%)	Harmful errors (%)	Fast-path share (%)	Latency (ms)		
						A100	L4	CPU
In-distribution	3.0	98.9	97.4	0.3	60.0	1.23	2.43	8.03
Cross-dataset	8.0	85.9	82.3	3.0	40.0	1.82	3.62	12.02
Paraphrased (no keywords)	10.0	91.1	86.3	2.0	5.0	2.85	5.70	19.00
Mixed-task	45.0	65.0	–	5.0	0.0	3.00	6.00	20.00
Out-of-distribution	55.0	–	–	18.0	0.0	3.00	6.00	20.00
Deployment mix (weighted)	9.1	–	–	1.7	43.0	1.73	3.44	11.42

Accuracy drops on cross-dataset and paraphrased requests, but the confidence threshold sends 8–10% of them to fallback and keeps harmful errors at 2–3%. For mixed-task and out-of-distribution requests, fallback rises to 45–55%, protecting accuracy at the cost of efficiency. Identification takes 1.2–3.0 ms per request on an A100 and 8–20 ms on a CPU.

End-to-end impact. Figure 19 runs LiteMoE with the classifier in the loop on each evaluation set. The optimistic and pessimistic cases account for fallback requests as defined in the fallback-rate analysis below. Throughput stays close to that of LiteMoE with oracle task labels for in-distribution, cross-dataset, paraphrased, and deployment-mix requests, and decreases most for mixed-task and out-of-distribution requests, where fallback is frequent. Accuracy changes by at most about 5 points in all cases.

Cost of harmful errors. Figure 20 isolates the cost of a harmful error for CONV, the most routing-sensitive task. Misclassifying CONV as QA lowers its cosine similarity from about 0.9 to 0.66–0.80, and misclassifying it as CLSFY/COMP lowers it to 0.46–0.62. Because such errors are rare in the deployment mix (1.7%), their expected effect on average accuracy is small.

Sensitivity to the fallback rate. Figure 21 varies the fraction of requests that fall back to standard expert loading from 0% to 40%. A fallback request loses task-aware selective loading and therefore runs like LiteMoE/L. How much this slows down other requests depends on batching, so we report two cases. In the *optimistic* case, only the fallback request loses selective loading; with fallback

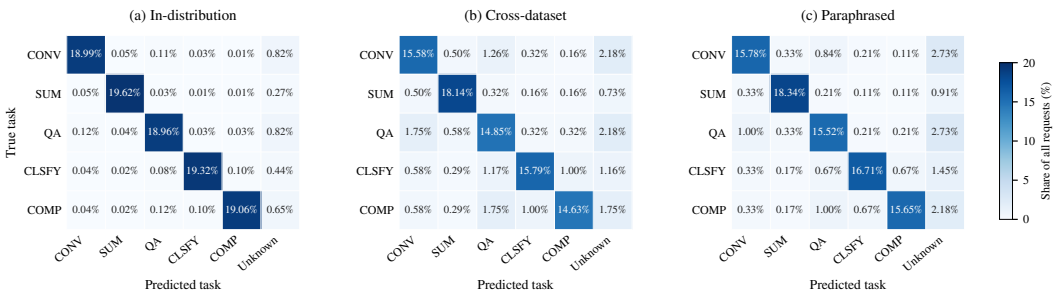
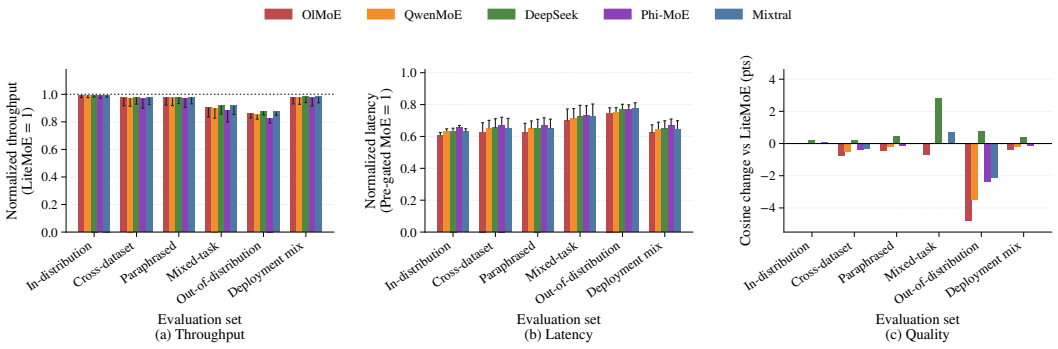
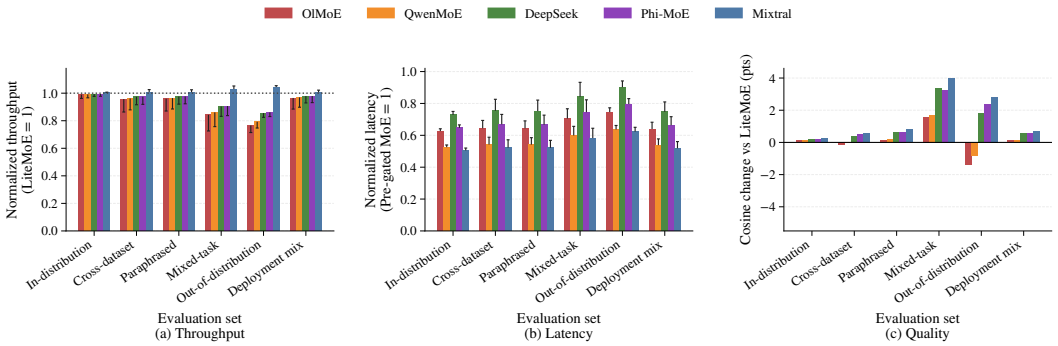


Figure 18: Task-identification confusion matrices for (a) in-distribution, (b) cross-dataset, and (c) paraphrased requests. Rows denote true tasks and columns denote predicted tasks; Unknown indicates fallback. Cell values are percentages of all requests, with each true-task row accounting for 20%.



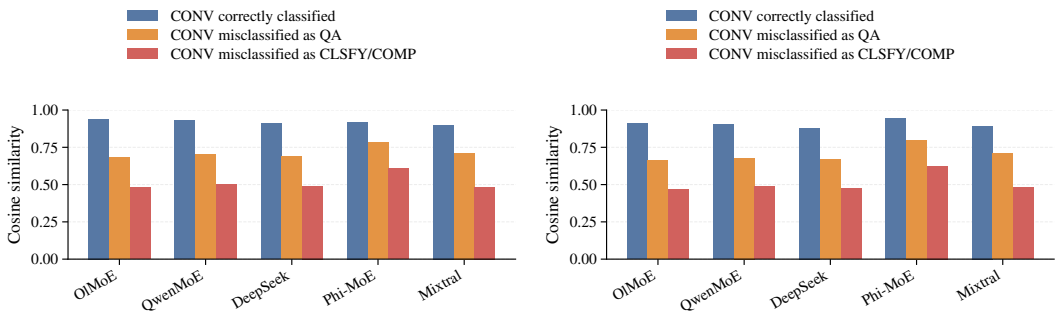
(i) A100



(ii) L4

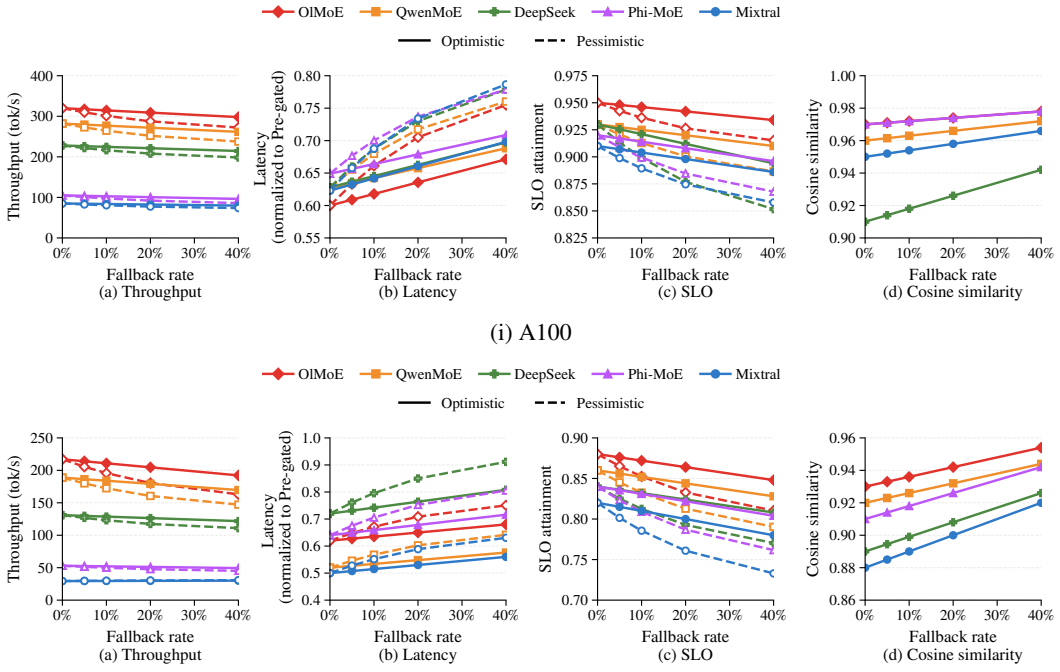
Figure 19: End-to-end impact of task classification on A100 (i) and L4 (ii). (a) Throughput is normalized to the corresponding model's LiteMoE throughput with oracle task labels; the dotted line marks this baseline (1.0). (b) Latency is normalized to Pre-gated MoE and includes classifier cost. (c) Change in cosine similarity relative to LiteMoE, in points. Bars show the optimistic case and whiskers extend to the pessimistic case (defined in Appendix A.5).

rate ϕ, performance is $(1-\phi)\cdot$LiteMoE $+\ \phi\cdot$LiteMoE/L. In the *pessimistic* case, a single fallback request in a batch of size b forces its predicted experts to be loaded for the whole batch, so every request in that batch loses selective loading; the affected share of batches is $p = 1-(1-\phi)^b$, and performance is $(1-p)\cdot$LiteMoE $+\ p\cdot$LiteMoE/L, with $b = 4$. The actual behavior lies between the two cases. Cosine similarity has a single value because it is determined per request: a fallback request always receives exact routing, and the other requests in its batch can only gain experts. Because fallback requests receive exact routing, cosine similarity increases with the fallback rate, while throughput, latency, and SLO attainment degrade gradually. Even at a 40% fallback rate, well



(a) A100 (b) L4

Figure 20: Effect of misclassifying CONV requests as QA or CLSFY/COMP on cosine similarity.



(i) A100

(ii) L4

Figure 21: Sensitivity to fallback rate on A100 (i) and L4 (ii). Latency is normalized to Pre-gated MoE. Cosine similarity rises with fallback because fallback requests receive exact routing. Solid lines show the optimistic case and dashed lines the pessimistic case (defined in Appendix A.5).

above the 9.1% observed on the deployment mix, LiteMoE retains lower latency than Pre-gated MoE on both GPUs.

A.6 Incremental Ablation

Figures 22 and 23 add LiteMoE's components one at a time on top of Pre-gated MoE: multi-layer prediction (with rollback), task-aware loading, and scheduling (full LiteMoE). Each gain below is relative to the preceding configuration. Table 3 reports the corresponding transfer operations and miss rates: multi-layer prediction with $B = 4$ issues one transfer per group of four MoE layers instead of one per layer, whereas task-aware loading and scheduling change which experts are transferred and when, but not the number of transfer operations.

Ablation on A100. Multi-layer prediction increases throughput by 7.7–24.4% and reduces latency by 2.3–5.2%, while improving SLO attainment by up to 2.0 percentage points. Adding task-aware

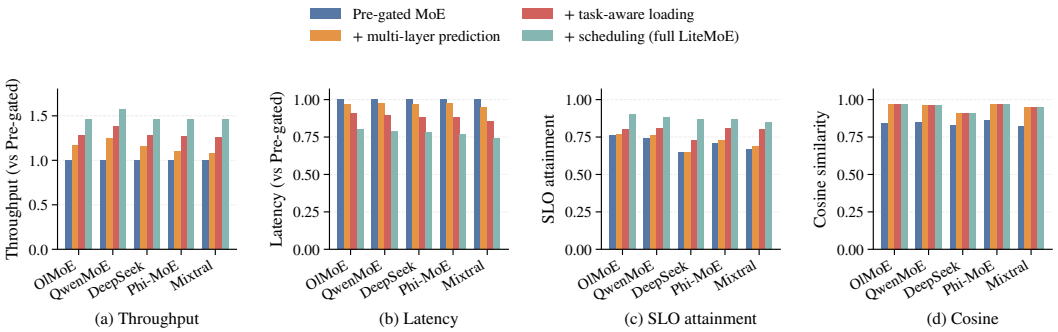


Figure 22: Incremental ablation on A100. The four configurations are Pre-gated MoE, + multi-layer prediction, + task-aware loading, and + scheduling (full LiteMoE). Throughput and latency are normalized to Pre-gated MoE.

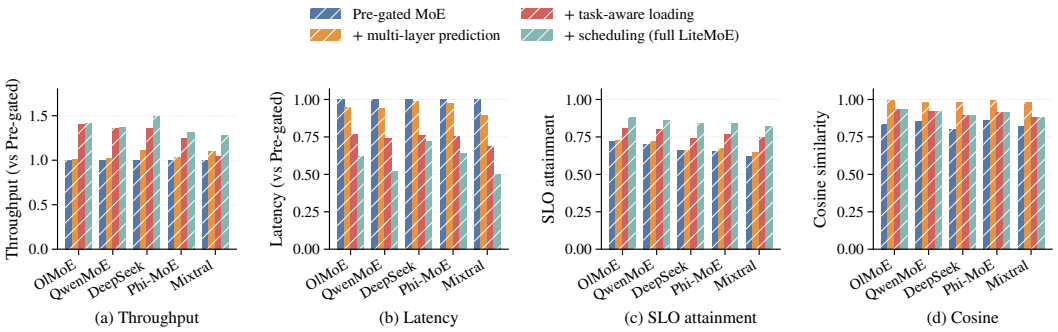


Figure 23: Incremental ablation on L4 (hatched bars). The four configurations are Pre-gated MoE, + multi-layer prediction, + task-aware loading, and + scheduling (full LiteMoE). Throughput and latency are normalized to Pre-gated MoE.

loading provides a further 9.7–16.6% throughput gain and a 5.8–9.4% latency reduction, together with a 3.0–11.0 percentage point increase in SLO attainment. Scheduling increases throughput by another 14.0–16.4%, reduces latency by 11.4–13.7%, and improves SLO attainment by 5.0–14.0 percentage points.

Ablation on L4. Multi-layer prediction increases throughput by 0.4–11.5% and reduces latency by 1.2–10.8%, with SLO attainment gains of up to 2.9 percentage points. Adding task-aware loading further reduces latency by 19.5–23.4% and improves SLO attainment by 8.0–10.0 percentage points across all five models. Its throughput benefit varies by model: OlMoE, QwenMoE, DeepSeek, and Phi-MoE improve by 20.5–40.1%, whereas Mixtral incurs a 5.2% decrease. Adding scheduling increases throughput by 0.3–22.2%, reduces latency by 4.8–29.6%, and improves SLO attainment by 6.2–10.0 percentage points.

A.7 Sensitivity Analysis

A.7.1 Sensitivity To Prediction Layer Group Size

LiteMoE uses a prediction layer group size of 4 to predict experts for the next four layers. Figure 24(a) illustrates the average router prediction accuracy across different group sizes for five models. It measures the fraction of router-selected experts that are correctly included in the predicted set. As the group size increases from 1 to 16, a gradual decline in prediction accuracy is observed across all models. For smaller group sizes, expert selection patterns are more stable and closely tied to the current input, making short-term predictions more reliable. In contrast, as the prediction horizon extends, the model must estimate experts without knowledge of future hidden states or token dynamics. This adds uncertainty, as small changes in intermediate computations can alter expert selections in later layers, reducing accuracy.

Table 3: Transfer operations per step and miss (rollback) rate (%) for the four incremental ablation configurations. Both metrics depend only on the model and predictor, and are identical on A100 and L4. S1: Pre-gated MoE; S2: + multi-layer prediction; S3: + task-aware loading; S4: + scheduling (full LiteMoE). A dash indicates that rollback is not applicable.

	Transfer ops / step				Miss rate (%)			
Model	S1	S2	S3	S4	S1	S2	S3	S4
OlMoE	16	4	4	4	–	13.0	13.0	13.0
QwenMoE	24	6	6	6	–	11.7	11.7	11.7
DeepSeek	26	7	7	7	–	10.8	10.8	10.8
Phi-MoE	32	8	8	8	–	6.6	6.6	6.6
Mixtral	32	8	8	8	–	7.8	7.8	7.8

Among the models, Phi-MoE and Mixtral achieve higher prediction accuracy across group sizes due to their smaller expert pools, which simplify selection and improve reliability. In contrast, QwenMoE and OlMoE, with much larger expert pools, show lower accuracy, as the greater number of candidates makes prediction more challenging, especially at larger group sizes. DeepSeek achieves high prediction accuracy at small group sizes, but its accuracy degrades noticeably as the group size increases, indicating greater sensitivity to longer prediction horizons. Overall, the figure highlights a trade-off between prediction horizon and accuracy. While larger group sizes reduce expert loading frequency and communication overhead, they make accurate prediction more difficult due to increased uncertainty over longer horizons.

Panels (b) and (c) quantify this trade-off end to end. Larger group sizes issue fewer transfer operations and therefore reduce latency relative to Pre-gated MoE, but most of the reduction is already obtained at a group size of 4, with little further gain at 8 and 16. Without rollback, cosine similarity at a group size of 4 stays close to the Pre-gated MoE reference, whereas at 16 it falls below the reference for most models. This supports the default of $B = 4$, with rollback recovering the remaining accuracy.

A.7.2 Effect of Caching Experts on System Throughput

In this experiment, we evaluate the effect of varying the percentage of cached experts in GPU memory, ranging from 25% to 75% in increments of 25%. Tokens whose top-ranked experts are unloaded are routed to the next-best available experts (e.g., second-best). Following Pre-gatedMoE, we consider three standard caching policies: Last-In First-Out (LIFO), Least Frequently Used (LFU), and Least Recently Used (LRU). These policies are applied when evicting experts to accommodate new ones during inference. We compare the impact of these caching strategies on both Pre-gatedMoE and LiteMoE.

The results for the Mixtral model are shown in Figure 25, where throughput values are normalized to the throughput of Pre-gatedMoE without any expert caching. As the figure indicates, increasing the cache size leads to significant improvements in throughput. Notably, LiteMoE consistently outperforms Pre-gatedMoE across all cache sizes and caching policies. Among the caching strategies, LiteMoE performs best under the LFU policy. Since LiteMoE incorporates task-awareness in its expert reloading decisions, it implicitly captures the recurrence patterns in token-to-expert routing, which aligns well with LFU's frequency-based eviction policy. As a result, LiteMoE experiences fewer cache evictions under LFU compared to LRU or LIFO. In contrast, Pre-gatedMoE shows similar performance across all three caching policies. Being prompt-agnostic, its expert reloading and eviction decisions are effectively random with respect to task types, leading to comparable behavior regardless of the caching policy employed.

A.7.3 Accuracy Sensitivity to Varying Expert Upload Percentage

In this experiment, we vary the percentage of experts uploaded from the predicted expert set. If a request's predicted expert is not uploaded, the request instead uses the highest-ranked local expert. We then measure the cosine similarity between the generated output and the output produced with full expert uploading. The results for the five models are shown in Figure 26. From the figure,

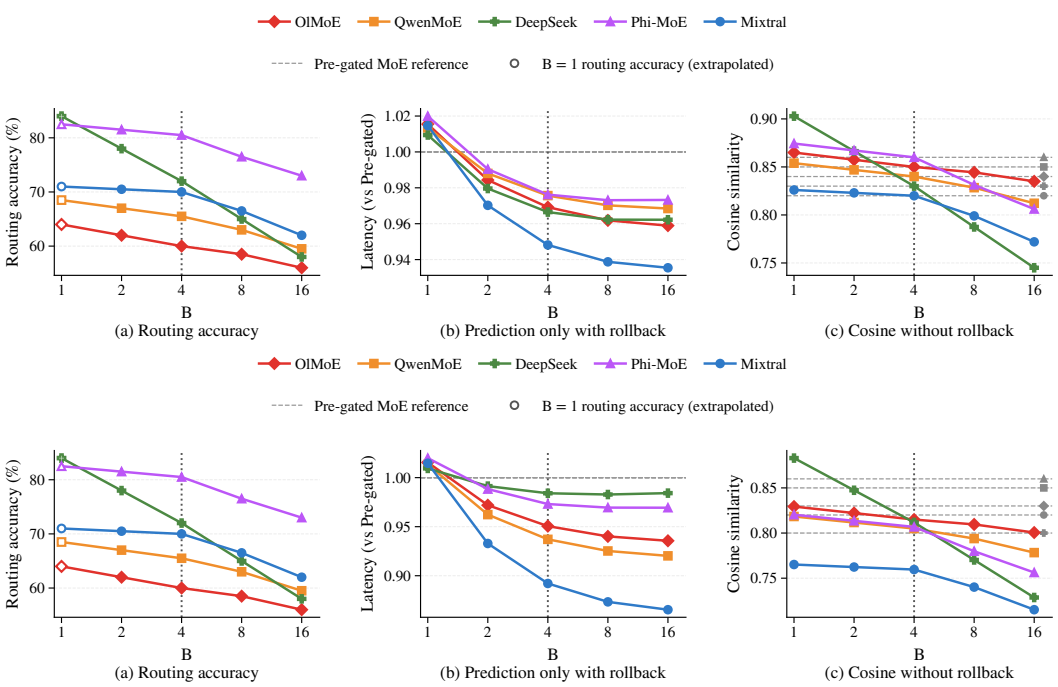


Figure 24: Sensitivity to prediction block size B on A100 (i) and L4 (ii): (a) routing accuracy, (b) normalized latency with prediction only and rollback enabled, and (c) cosine similarity without rollback. Hollow markers at $B = 1$ in (a) denote extrapolated routing accuracy. Horizontal gray dashed lines in (b) and (c) show each model's Pre-gated MoE reference: all latency references equal 1 and overlap; cosine references on A100 are 0.84, 0.85, 0.83, 0.86, and 0.82, in the order OlMoE, QwenMoE, DeepSeek, Phi-MoE, and Mixtral. Gray markers at the right edge of (c) identify reference models using the same marker shapes. For B = 1, 2, 4, 8, and 16, transfer operations per step are (16, 8, 4, 2, 1) for OlMoE, (24, 12, 6, 3, 2) for QwenMoE, (26, 13, 7, 4, 2) for DeepSeek, and (32, 16, 8, 4, 2) for both Phi-MoE and Mixtral.

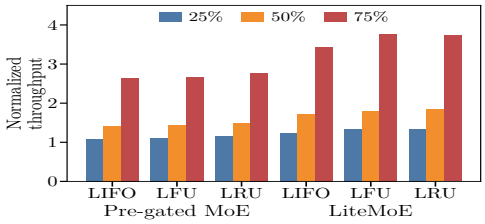


Figure 25: Effect of caching experts on throughput.

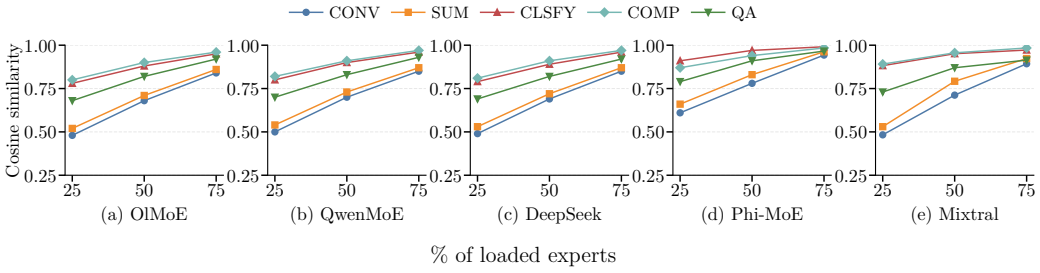


Figure 26: Impact of expert upload percentage on accuracy.

we observe that increasing expert upload percentage consistently improves output accuracy across all tasks. Notably, the improvement is most pronounced for conversation and summarization tasks. In contrast, simpler tasks such as classification or question answering exhibit lower sensitivity to expert upload percentage. Nevertheless, a significant accuracy degradation is observed when expert upload percentage drops below 50%. For open-ended tasks, however, the accuracy remains close to that of the fully utilized setting when the expert upload percentage is at or above 75%. This trend is consistent with the findings reported in §2.3.