

OD-MoE: On-Demand Expert Loading for Cacheless Edge-Distributed MoE Inference

Liujianfu Wang[†] Yuyang Du^{†,‡} Yuchen Pan Soung Chang Liew* Jiacheng Liu Kexin Chen

The Chinese University of Hong Kong

Abstract

Mixture-of-Experts (MoE), while offering significant advantages as a Large Language Model (LLM) architecture, faces substantial challenges when deployed on low-cost edge devices with tight memory constraints. Expert offloading mitigates this issue by storing expert parameters in CPU memory and caching a subset of popular experts in GPU memory. Although this approach improves GPU memory utilization by caching only the likely-used experts, the GPU memory reserved for expert caching is underutilized compared with dense LLMs. This paper presents **OD-MoE**, a distributed MoE inference framework that obviates the need for expert caches via fully **on-demand** expert loading. OD-MoE is built upon two key mechanisms: 1) parallelizing expert loading and expert computation across distributed edge nodes, and 2) an ultra-accurate emulative predictor that forecasts expert activations multiple layers ahead while expert computation is ongoing. With these innovations, OD-MoE dynamically loads each target expert to one of the distributed nodes just-in-time before its activation and promptly evicts it afterward, freeing GPU memory for subsequent experts. We comprehensively benchmark OD-MoE against state-of-the-art MoE offloading systems on a ten-node testbed. Experimental results show that: 1) OD-MoE achieves 99.94% expert activation prediction accuracy, substantially surpassing all existing methods; and 2) OD-MoE delivers approximately 75% of the decoding speed of a fully GPU-cached MoE deployment while using only 1/3 of the GPU memory. More importantly, by eliminating the need for expert caches, OD-MoE enables MoE inference on edge nodes with less-than-1GB GPU memory, paving the way for practical MoE deployment of low-cost IoT devices at the edge in the LLM era.

1 Introduction

The rapid explosion of Large Language Models (LLMs) has led to their widespread application across various fields

[6, 13, 14]. Beyond deploying LLMs in data centers and accessing them via cloud services, there is a growing demand to bring these models to edge. This shift aims to address challenges such as privacy concerns and the dependence on stable, wide-area network connections, both of which arise from centralized, data-center-based approaches [8, 21, 35]. By moving inference closer to where applications are deployed, edge deployment reduces latency, improves reliability, and alleviates privacy concerns by minimizing the need to transmit sensitive data over wide-area networks. Consequently, efficient LLM inference at the edge has emerged as an active area of research in both academia [9, 36, 40] and industry [1, 2, 26].

In recent years, the Mixture of Experts (MoE) architecture has emerged as a significant paradigm for scaling LLMs. Unlike dense models that activate all parameters for every input, MoE models selectively activate a small subset of experts during inference, thus enabling significant model size expansion while maintaining per-request computational efficiency [23, 28]. Yet, MoE models demand a substantial memory footprint, requiring 4-5× more GPU memory than dense models with comparable inference FLOPs [5]. This substantial memory requirement has become a major barrier for the deployment of MoE models on resource-constrained edge.

Expert offloading is a technique to address this memory challenge [5, 15, 21, 31, 34, 37, 38]. This method leverages the sparse activation pattern of MoE by storing the majority of expert parameters in slower yet more abundant CPU memory¹ and loading them into GPU memory only when required for computation. However, due to the limited bandwidth of the CPU-GPU link, loading an expert from CPU to GPU introduces significant latency, especially for edge devices that typically rely on standard PCIe buses rather than high-speed interconnects in data centers.

Existing edge inference systems have attempted to address the I/O bottleneck with various methods. For example, Edge-

[†]Equal contribution. [‡]Project lead. *Corresponding author.

¹In this paper, CPU memory refers to the DRAM of a device. We do not use SSD/NVMe storage due to their large access delays. We also do not rely on on-chip CPU caches (L1/L2/L3) because their capacities are negligible compared with an expert’s parameter size.

MoE, HOBBIT, and Mixtral-Offloading [15, 31, 37] employ quantization techniques to compress expert parameters; and AdapMoE [42] skips certain experts to alleviate the I/O bottleneck. However, these methods may cause significant degradation in model performance, especially when an important expert is highly compressed or skipped.

The I/O pressure between CPU and GPU can also be mitigated by caching likely-to-be-used experts in GPU memory. For example, Mixtral-Offloading [15] chooses to cache most recently used experts, while MoE-Infinity [34] caches most frequently used ones. HOBBIT [31] quantizes experts into different precision levels and prefers keeping high-precision ones in the GPU cache. Beyond GPU memory needed for the ongoing expert computation, these approaches require additional GPU memory for expert caching. The additional memory requirement can potentially limit the edge system’s capacity in hosting large-scale MoE models.

This paper presents **OD-MoE**, a distributed MoE inference framework that eliminates the need for expert caching achieves by enabling fully **On-Demand** expert loading. **OD-MoE achieves 75% of the decoding speed of an MoE deployment with all experts cached in GPU memory, yet requires only 1/3 of the GPU memory without compromising model performance via methods like quantization or expert skipping.** We refer readers to Table 2 in Section 4 for detailed experimental results about the GPU memory requirement and a comprehensive speed evaluation.

Our key contributions, along with the underlying design principles of OD-MoE, are summarized as follows:

Our first contribution is an expert-activation predictor with an accuracy up to 99.94%, which, to the best of our knowledge, is the highest accuracy reported to date. The predictor is inspired by LLM quantization techniques [17, 25]. With model quantization, the compressed model runs faster and requires less GPU memory, but exhibits highly similar behavior to the original full-precision model, including expert routing. Building on this idea, our method diverges from approaches that predict the next layer’s experts based on the current layer’s state [15, 31, 34, 37, 38]. Instead, we employ a low-cost, quantized MoE model, referred to as the “shadow” model, which runs in parallel with the full-precision model. The shadow model acts as a faster-running emulator to predict the expert activations of the full-precision model several layers ahead. Specifically, this predictor uses the future expert activations that are already unfolded by the scaled-down shadow model to forecast the expert activations of the full model. We refer to this approach as **Scaled Emulative Prediction (SEP)**.

Our second contribution is the elimination of the need for an expert cache by leveraging the ultra-accurate SEP and parallel expert loading-computation over distributed edge nodes. OD-MoE consists of multiple groups of low-cost edge nodes, each with its own CPU-GPU interconnect. Thanks to the ultra-accurate lookahead predictions provided by SEP, distributed devices are fully aware of the expert activations for

subsequent layers. This allows one group of nodes to perform expert computation for the current layer while other groups simultaneously load the experts needed for upcoming layers based on the predicted activations. Through cross-device parallelism in expert loading, OD-MoE loads a target expert to one of the distributed nodes just-in-time before its activation and promptly evicts it after computation is complete, making room for subsequent expert loading into GPU memory. From a prospective of I/O bandwidth utilization, the key enablers of OD-MoE’s cache-free inference are 1) the significantly increased overall CPU-GPU I/O throughput achieved through parallel loading across devices, and 2) minimal I/O bandwidth waste, as SEP’s highly accurate predictions rarely result in incorrect expert loads that trigger reloads.

Our third contribution is the development of two essential alignment mechanisms within SEP: KV cache alignment and token alignment. During the shadow model’s decoding process, it generates new tokens and KV cache autoregressively. Although the shadow model closely mimics the behavior of the full-precision model, differences in precision levels can occasionally result in varying outputs, causing discrepancies in the generated KV caches and tokens. While SEP is highly reliable at the beginning of the inference, its accuracy decreases gradually as discrepancies accumulate in the autoregression process (see Fig. 3). Periodically aligning the shadow model’s tokens and KV cache with those of the full-precision model helps prevent the cumulative propagation of errors from one round to the next. While the idea is straightforward, its implementation requires navigating a nontrivial trade-off between prediction accuracy and alignment cost. Alignment must be performed before the shadow model begins generating the next token, even as the full-precision model’s computation is already underway, resulting in a delayed start for the shadow model. Although the shadow model quickly catches up after alignment, predictions for the first few layers are temporarily unavailable, forcing the inference system to revert to an I/O-bottlenecked state for these layers (see Fig. 5).

Overall, OD-MoE offers several benefits:

1. **Reduced GPU Memory Requirements:** By eliminating the need for an expert cache, OD-MoE significantly reduces GPU memory usage on edge devices. For instance, in our implementation based on Mixtral-8×7B [20], the GPU memory footprint per worker node is less than 1 GB, including space for the currently scheduled expert and necessary compute memory. This reduction allows even low-cost devices, such as Wi-Fi routers or webcams, to participate in inference tasks, prompting a rethinking of the role of low-cost, underutilized IoT devices in everyday environments. These devices can now become active contributors to LLM inference systems.
2. **Lower Hardware Costs:** OD-MoE requires only one-third of the GPU memory needed for a fully GPU-cached deployment, reducing hardware costs by over threefold.

As discussed earlier, OD-MoE leverages low-cost edge GPUs, which are significantly cheaper than data center-grade GPUs due to their smaller memory capacity. This enables cost-effective utilization of entry-level GPUs, which have a lower per-GB memory cost compared to advanced GPUs [12].

3. **Applicability to Data Center Deployments:** While OD-MoE is primarily designed for edge scenarios, its SEP scheme and parallel expert loading mechanism can also benefit data center operations. For instance, accurate predictions of future expert usage can serve as the foundation for on-demand expert replication within a cluster – a proven method for mitigating server workload imbalances [18]. Additionally, OD-MoE’s cost-effective approach, which uses less expensive GPUs with smaller memory capacities and efficiently offloads experts without being bottlenecked by GPU I/O, may also be advantageous for data centers.

We have open-sourced OD-MoE. The released resources include project implementations, testing scripts, and a comprehensive benchmarking report that compares our model with previously reported methods for both expert-activation prediction and expert offloading. The report also provides detailed implementations of the baseline methods to ensure reproducibility and fair comparisons. The project is available at <https://github.com/Anonymous/DoubleBlind>.²

2 Related Work

2.1 Edge MoE Inference with All Experts Cached

Distributed inference has emerged as a promising research direction for LLM serving systems utilizing MoE models. The architecture of MoE facilitates model deployment across multiple interconnected nodes, where experts can be deployed and activated according to traffic patterns, network conditions, user demands, and device workloads. Several recent works on distributed MoE inference assume that all experts are pre-loaded into GPU memory, eliminating the need for dynamic expert loading. These works primarily focus on optimizing expert placement to maximize inference throughput.

For example, CoEL [21] introduces a collaborative inference framework for expert placement across memory-constrained devices. SlimCaching [8] formulates expert placement on edge devices as a knapsack problem (KP) to derive the optimal static caching policy for edge nodes. WD-MoE [35] considers the influence of wireless channel when placing experts – edge nodes with good channel conditions are preferred when placing frequently used experts since these devices lower inter-node communication delays.

²The link will be made public upon publication to comply with the double-blind review process.

In contrast to these approaches, which assume all expert parameters are pre-cached in GPUs, **OD-MoE** eliminates the need for any pre-cached experts by relying on dynamic expert loading. This approach significantly reduces GPU memory footprints, enabling practical MoE deployments on low-cost edge devices.

2.2 Dynamic Expert Loading and I/O Utilization

As an MoE model activates only a subset of experts during inference and leaves a large portion of experts inactivated, expert offloading offers a cost-effective solution for running MoE models with limited GPU memory. A general principle of expert offloading is storing less popular experts within CPU memory and dynamically loading them into the GPU cache only if they are wanted during model inference.

However, long expert loading times pose a significant challenge, particularly due to the limited GPU I/O bandwidth on edge devices that rely on PCIe buses for CPU-GPU communication. To alleviate computation stalls during expert loading, recent works have proposed various expert caching strategies to reduce the volume of dynamic expert loading required. Mixtral-Offloading [15] and AdapMoE [42] manage expert cache pools by offloading the least recently used (LRU) experts, whereas MoE-Infinity [34] evicts the least frequently used (LFU) experts when the expert cache is full. Furthermore, fMoE [38] incorporates a semantic matching scheme between historical prompt and the current input into the cache management scheme to increase its cache-hit rate. Building on these classical caching strategies, HOBBIT [31] and EdgeMoE [37] further take the LLMs’ model-specific features into consideration. For example, HOBBIT, which quantizes experts into different precision levels, prioritizes retaining frequently used high-precision experts when the cache is full. Meanwhile, EdgeMoE favors evicting experts from layers farther away from the one currently being processed.

Another approach to mitigate the computation stalls during expert loading is to sacrifice the model’s performance through expert quantization. This technique, applied in Mixtral-Offloading, AdapMoE, Hobbit, and EdgeMoE [15, 31, 37, 42], reduces the volume of data to be loaded. Additionally, AdapMoE [42] introduces a bypass mechanism to skip activating experts that are not cached.

OD-MoE differs from the above work with its fully on-demand expert loading mechanism, which eliminates the need for any expert cache in GPU memory while maintaining both full model precision and all expert activations.

2.3 Expert-Activation Prediction

Accurate expert-activation prediction is the foundation for dynamic expert loading. Prior systems have explored various

prediction methods, with two major research directions being investigated.

The first approach relies on statistical models to predict expert activation. For instance, [37, 38] build statistical models based on each expert’s historical expert-activation frequencies (i.e., the popularity of this expert) to predict the most likely expert selection. MoE-Infinity [34] further improves this approach by recording each expert’s popularity under different requests – it matches the given prompt to one of those recorded requests and makes expert-activation predictions based on the historical data associated with that request.

The second approach predicts expert activation on the fly. For example, [15, 41, 42] adopt simple heuristic: while the latest embedding vector is fed to the current gating network for expert selection, it is also fed to the subsequent gating network for predicting the next layer’s expert activation. HOBBIT [31] improves upon this method by aggregating the gating networks of successive layers into a larger gating network with multiple layers so that it can predict the expert activation for successive layers simultaneously.

While these prediction methods have been proven to be effective, an even more precise predictor is highly desirable due to the severe penalty for mispredictions. Experts that are incorrectly preloaded must be evicted and replaced with the correct ones, causing significant system stalls as the planned expert activation must wait for the reloading process to complete.

Our prediction method, **SEP**, belongs to the on-the-fly prediction category but differs significantly from previous methods. At its core, SEP performs an “emulation” of the full model’s expert activation. Specifically, SEP uses the future expert activations unfolded by a faster shadow model to predict the expert activations of the full-precision model, resulting in naturally more reliable predictions. Moreover, since the shadow model runs faster than the full model, SEP provides predictions several layers ahead of the target expert activation.

With FP16 quantization applied to the shadow model, SEP achieves an average prediction accuracy of 99.94% across multiple layers. Even when more economical INT8 and NF4 quantization are used, SEP maintains prediction accuracies of 97.34% and 95.67%, respectively, outperforming the state-of-the-art prediction accuracies reported in related works.

3 System Design

Subsection 3.1 presents the distributed architectural design of OD-MoE, focusing on expert distribution, as well as the scheduling of computations and loading among nodes for the decoding stage. Subsection 3.2 delves into the technical details of SEP alignment operations. Subsection 3.3 describes the operation of the prefilling stage in OD-MoE, which requires a different distribution of expert computations across nodes compared to the decoding stage to achieve optimal performance.

3.1 OD-MoE Architecture and Decoding-Stage Scheduling

Fig. 1 illustrates the system architecture of OD-MoE using the ten-node testbed we developed. OD-MoE consists of three types of nodes: 1) a **main node** hosting all non-expert components in the MoE model, including attention networks, gating networks, normalization networks, and others; 2) a **shadow node** runs SEP in parallel with the main node to predict the expert activation and notify the worker nodes accordingly; and 3) **worker nodes** that dynamically load experts into the GPU based on prediction and execute the associated expert computations.

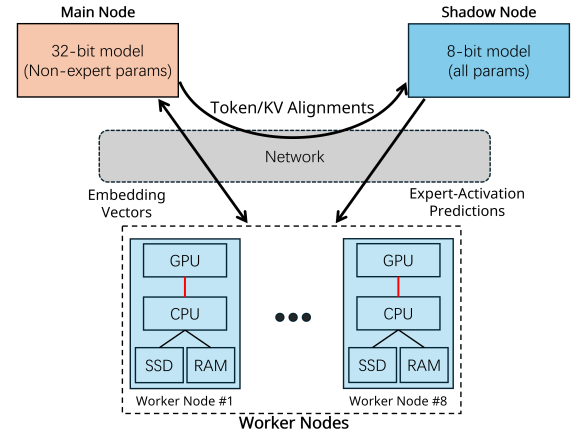


Figure 1: Architecture of OD-MoE. The example here shows the ten-node testbed we have developed, which includes eight worker nodes, one main node, and one shadow node.

Key methodologies in OD-MoE include worker-node grouping and round-robin scheduling. These concepts are introduced alongside the timing diagram shown in Fig. 2.

Worker-Node Grouping. With the several-layer-ahead predictions provided by the shadow node, OD-MoE employs a “group-and-schedule” strategy to orchestrate the workload of distributed workers. This strategy aims to mitigate the GPU I/O pressure by parallelizing expert activations (handled by one group of devices) and on-demand expert loadings (handled by other groups of devices). Specifically, worker nodes are divided into N_W/G groups, where N_W is the number of workers (eight in our testbed) and G is the group size. All workers within a group either load or execute experts in parallel for a particular MoE layer in a synchronous manner. In our implementation, $G = 2$ since a top-2 activation policy is applied in the Mixtral-8x7B model. Each expert is loaded to only one worker node. This one-to-one assignment of experts to workers prevents uneven expert distributions that could overload specific worker nodes.

Round-Robin Scheduling. As illustrated in Fig. 2, MoE inference for successive layers is scheduled across different groups of devices in a round-robin manner. That is,

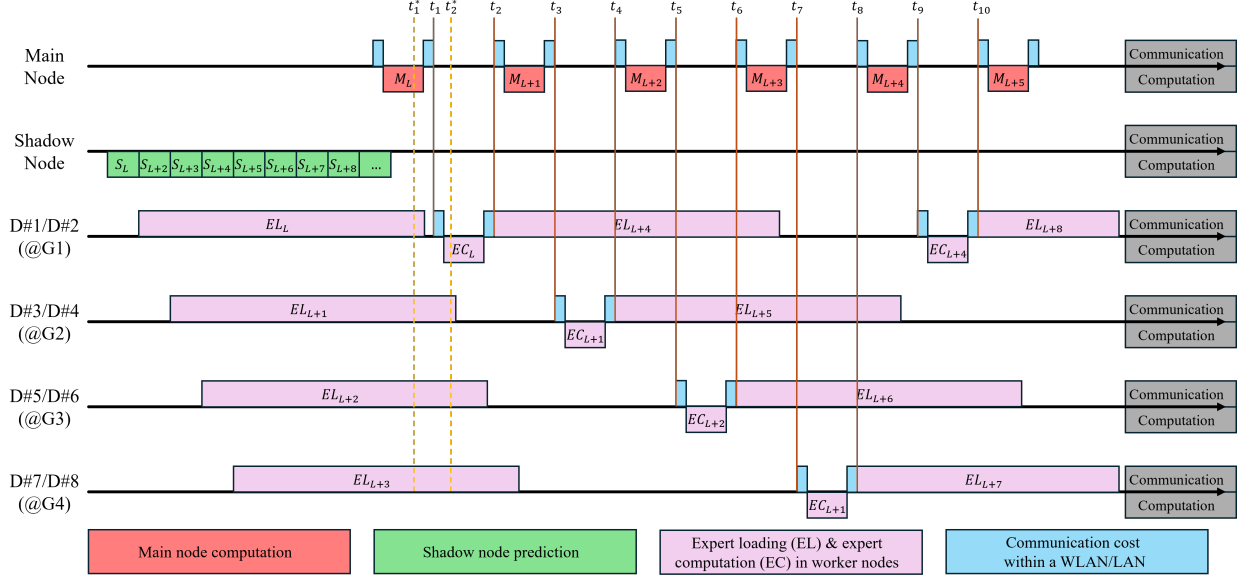


Figure 2: Timing diagram of OD-MoE illustrating the round-robin scheduling scheme therein. Main node computation for layer l is denoted by M_l . Shadow node computation for layer l is denoted by S_l . Expert loading and expert computation for layer l are denoted by EL_l and EC_l

- While devices in Group 1 perform expert computation EC_l from t_1 to t_2 , devices in Group 2, Group 3, and Group 4 simultaneously perform expert loading for future layers EL_{l+1} , EL_{l+2} , and EL_{l+3} respectively, in a staggered manner.
- Once EC_l completes at t_2 , devices in Group 1 (Devices #1 and #2) shift to expert loading task EL_{l+4} to get ready for future expert computation EC_{l+4} .
- Devices #1 and #2 send embedding generated in EC_l to the main node, and the main node receives the embedding after a communication delay (the tiny blue segments in Fig. 2). The main node performs its non-expert computations M_{l+1} , including attention, gating, and normalization.
- Once M_{l+1} completes at t_3 , the main node sends the embedding vector to Group 2 (Devices #3 and #4) for processing at the next layer, with a similar communication delay.
- Operations after t_3 mirror Step *a* to Step *d* above. The pipeline repeats until computations for all layers are done.

We highlight several important technical details in the pipeline. First, the communication time in Steps *c* and *d* in the above includes the time spent on the following components: 1) converting the embedding vector into a packet, 2) delivering the packet over the network (e.g., over WLAN/LAN), 3) converting the packet back into the embedding vector for GPU computation.

Second, we note that a gating network is part of the non-expert parameters stored in the main node. During a main-node computation task, such as M_{l+1} in Step *d* above, the

gating network is activated as in a standard MoE system to determine expert routing for the current layer. If the routing results agree with the predictions generated by the shadow node, no additional expert loading is necessary (e.g., by t_3 , all predicted experts will have already been loaded into Devices #3 and #4). In rare instances where prediction errors occur, the pipeline falls back to the conventional approach, where the expert computation task waits for the completion of expert reloading. However, since such cases are infrequent, their impact on the system’s overall inference speed is minimal.

Third, as shown in Fig. 2: 1) when the main node performs its computation task, up to N_W devices (i.e., all devices) can work concurrently to load experts (see t_1^* in Fig. 2); 2) when a group of worker nodes is busy with expert computations, the remaining $N_W - k$ devices load experts concurrently (see t_2^* in Fig. 2). In essence, the former increases the effective throughput of the system’s CPU-GPU I/O link by N_W -fold, while the latter boosts the effective throughput by $(N_W - k)$ -fold. For easier identification, critical CPU-GPU links are highlighted in red in Fig. 2. The multiplied throughput is a key advantage of OD-MoE over expert-offloading systems with a single node (see Section 2 for related works).

Finally, we provide the timing requirement for fully eliminating the CPU-GPU I/O bottleneck, assuming correct expert-activation prediction. Suppose that the main node requires t^M to complete the main-node computation task (including the communication overhead), i.e., $t^M = t_3 - t_2 = t_5 - t_4 = \dots$.

Similarly, suppose that a worker node requires t^W to complete the expert computation task (also including the communication overhead), i.e., $t^W = t_2 - t_1 = t_4 - t_3 = \dots$.

Then, the maximum allowable duration allowed for the expert loading task without introducing an I/O bottleneck, as shown in Fig. 2, is

$$t^{\maxload} = Gt^M + (G-1)t^W \quad (1)$$

For example, for expert loading task EL_{l+4} starting at t_2 , there is no compute stall for EC_{l+4} waiting for expert loading to complete if the expert loading completes before t_9 , i.e., $t^{\maxload}(EL_{l+4}) = t_9 - t_2 = 4t^M + 3t^W$.

In practice, since the parameter size of an expert and the available throughput of a device’s CPU-GPU link are known, we can calculate the required expert loading time and compare it with $t^{\maxload}$ to determine whether the system is I/O-bottlenecked.

3.2 Token and KV Cache Alignments in Shadow Model

Model quantization is a well-established technique originally designed to reduce the memory footprint and computational cost of LLM inference, often with only modest degradation in the quality of the LLM’s responses. Consequently, when a quantized model and its full-precision counterpart process identical inputs, their internal states—such as embedding vectors, attention scores, and output logits—are expected to be highly similar. In particular, the expert selection pattern of a quantized model closely mirrors that of the full-precision model for the same input.

We have experimentally observed this phenomenon during the generation of the first token in the decoding stage. However, as additional tokens are generated autoregressively, the prediction accuracy of SEP deteriorates rapidly (see blue curves in Fig. 3). This degradation is the result of small numerical errors introduced by quantization, which compound during the autoregressive token generation process. The divergence in the shadow model’s expert selection can be attributed to two sources: 1) **Divergent Generation Paths**: the quantized model may generate a different token than the full-precision model. 2) **Internal KV-State Drift from Routing Differences**: differences in expert selections between the quantized and full-precision models can cause their embeddings to diverge, resulting in discrepancies in their KV cache.

A direct approach to restoring accurate expert-activation prediction in SEP is to align the shadow model’s full KV caches of all attention heads of all decoding layers and tokens with those of the full-precision model to prevent error accumulation. Fig. 3 presents experimental results showing the effectiveness of this approach. The results of different alignment set-ups are shown: 1) unaligned shadow model, 2) token aligned after each autoregressive iteration, 3) both token and KV cache aligned after each autoregressive iteration.

Fig. 3 shows the prediction accuracy of SEP across various different alignment setups. We follow previous works [31,

41, 42] and evaluate prediction accuracy using the recall rate. The recall rate is defined as the number of correctly predicted experts out of all activated experts.

To be precise, consider an MoE model with L layers that employs a top- k activation policy. Let Q be the number of test prompts and N be the maximum number of decoding iterations (i.e., decoded output tokens) in our experiments. We terminate the decoding stage of a prompt when the number of tokens exceeds N . Also, the decoding of some of the Q prompts may end before output token N . To capture this, let $A(q, n) \in \{0, 1\}$ be an indicator for the existence of token $n \in [1, N]$ for prompt $q \in [1, Q]$. For expert-activation predictions, let $c(q, n, l) \in [0, k]$ be the number of correctly predicted experts for layer $l \in [1, L]$, iteration n , and prompt q .

With the above, the average recall rate for output token n in the decoding stage is defined as

$$\text{recall}(n) = \sum_{q=1}^Q \sum_{l=1}^L c(q, n, l) A(q, n) / kL \sum_{q=1}^Q A(q, n) \quad (2)$$

and the overall recall rate over the tokens observed is given by

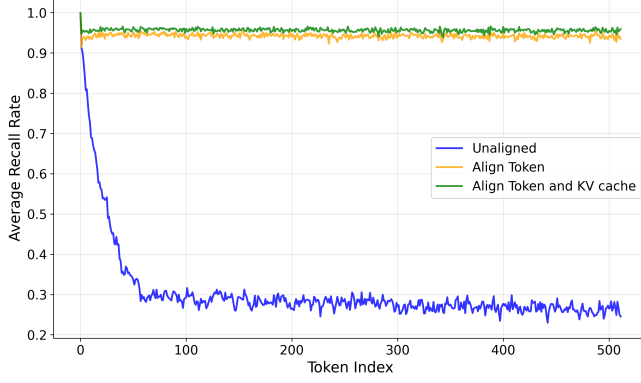
$$\text{recall} = \sum_{n=1}^N \sum_{q=1}^Q \sum_{l=1}^L c(q, n, l) A(q, n) / kL \sum_{n=1}^N \sum_{q=1}^Q A(q, n) \quad (3)$$

In this paper, $k = 2$ and $L = 32$ for the Mixtral-8x7B model tested. We selected $Q = 100$ test problems randomly from the LongWriter dataset [4] and set $N = 512$ for experimental observations.

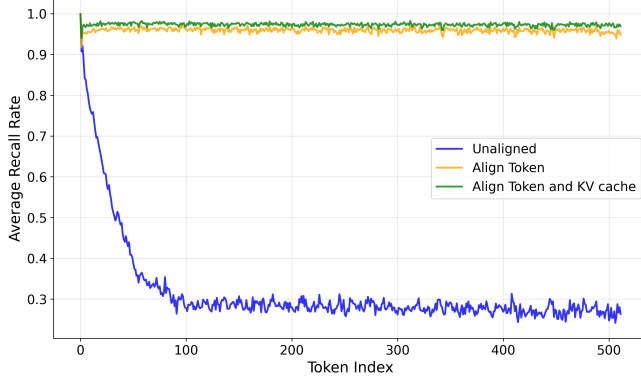
Fig. 3 presents the recall rate versus token index for the sequence of tokens generated during the decoding stage. As shown, both KV cache alignment and token alignment play a vital role in ensuring the prediction accuracy. By synchronizing tokens and the KV cache for each autoregressive iteration during inference, SEP yields approximately 99.94%, 97.34%, and 95.67% recall rates given by (3), with FP16, INT8, and NF4 quantization in the shadow model, respectively. In Section 4, we will show that even the NF4-quantized model beats state-of-the-art baselines.

The next issue we need to consider is the implementation of KV cache and token alignment. Frequent KV cache and token alignments introduce additional latency referred to as the “late-departure cost”. To better understand that timing cost, we start with a normal timing diagram in Fig. 4 to illustrate how the shadow model starts the inference without KV or token alignment. After that, we present the timing diagram in Fig. 5 to aid visualization on how alignments cause the shadow model’s “late departure”, and what cost it introduces.

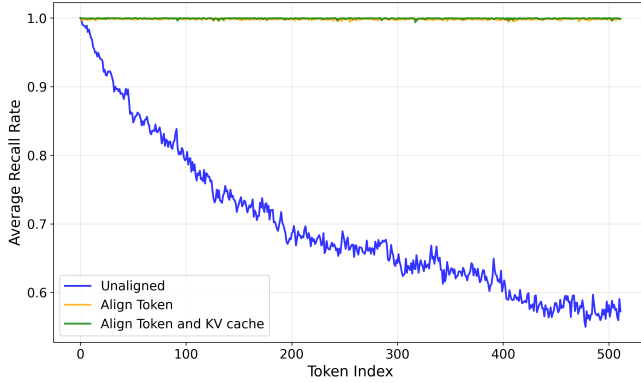
In Fig. 4, both the shadow node and the main node kick off the inference for the first layer at t_0 . The shadow node runs faster and provides its routing results as a reference at t_1 , which triggers the expert loading task EL_1 for Devices #1 and #2. Although M_1 ends shortly at t_2 , the associated expert



(a) NF4



(b) INT8



(c) FP16

Figure 3: Expert-selection recall rate versus output token index. Three different quantization schemes (NF4, INT8, and FP16) have been considered for the shadow model, while the original model is realized with a precision of FP32.

computation task EC_1 has to wait for the completion of EL_1 at t_4 . As indicated in Fig. 4, an I/O bottleneck appears during the inference of the first MoE layer. Subsequent layers avoid I/O stalls. Task EL_2 starts on Devices #3 and #4 as early as t_3 . By the time task M_2 ends, expert needed for layer 2 have been loaded into the two workers so that the task EC_2 does

not need to wait as EC_1 does.

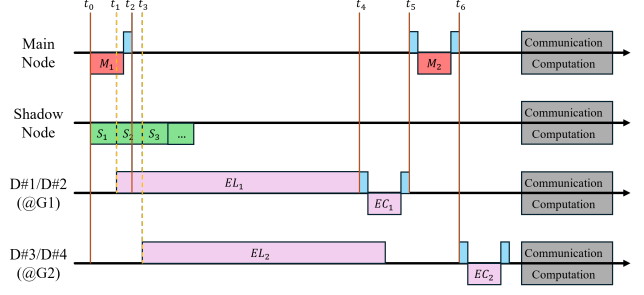


Figure 4: Illustrative timing diagram of OD-MoE, without alignment for the shadow model.

To reduce the negative impact of alignments on decoding latency (see our later discussions), we could align the tokens and KVs once every few autoregression iterations, at the cost of a lower recall rate. We note that the alignment periods for tokens and KVs could be different. The details of the impacts of alignment periods on decoding speed from experimental results will be presented Section 4. Here in Section 3, we explain how alignments can lead to increased decoding latency by timing diagrams. We also present some results on the impacts of alignment frequencies on the recall rate.

Given that 1) the most up-to-date KV cache and tokens of the original full-precision model are available only when the current autoregressive iteration ends, and 2) the main node moves on to the next iteration immediately after the completion of the current iteration, we can use the beginning part of the new iteration to align the shadow model with the latest KV cache and token generated by the full model. That leads to a shadow model’s “late departure” problem as illustrated in Fig. 5.

In Fig. 5, expert loading task EL_1 relies on the routing results of the main model available at t_1 . The routing prediction for layer 2 is not available until the shadow model catches up at t_3 . Although EL_2 starts immediately on Devices #3 and #4 at t_3 , the loading task has not yet completed by the time M_2 ends at t_7 . Thus, the I/O bottleneck persists in layer 2, keeping

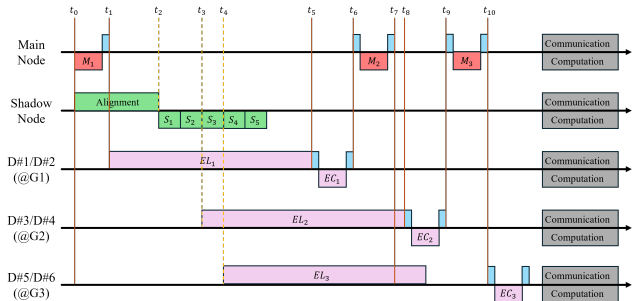


Figure 5: Illustrative timing diagram of OD-MoE, with alignment for the shadow model.

expert computation task EC_2 waiting until EL_2 ends at t_8 .

Comparing Fig. 4 and Fig. 5, we see that the late departure of the shadow model, caused by the alignment operation, prolongs the I/O bottleneck, thus adding to the decoding latency. Fig. 6 presents the experimental results about the relationship between recall rate and token/KV alignment periods.

We see that there is a clear trade-off we need to consider between the “late-departure cost” and the reduced accuracy in terms of expert-activation prediction. Section 4 experimentally investigates this balancing required to obtain the optimal setup for the alignment operations.

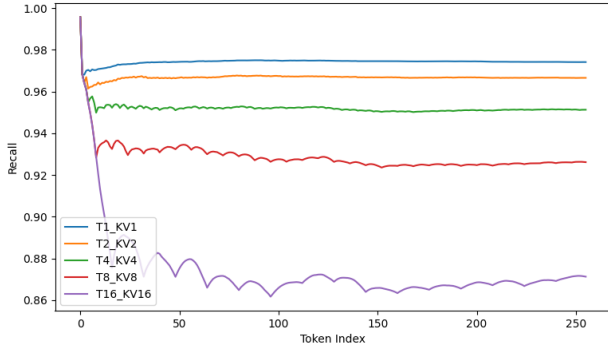


Figure 6: Average expert activation prediction accuracies for token and KV cache alignment intervals of 1/2/4/8/16. In the legend, $T_i_KV_j$ denotes token alignment period of i and KV alignment period of j . An INT8 quantized shadow model is used, reaching over 97.34% prediction accuracy with the $T1_KV1$ setup.

3.3 Prefilling-Stage Batched Processing

In prefilling, the entire input prompt is processed in a single batch to create the initial attention KV cache, which is used to generate the first output token in the decoding stage. With batched execution, token embeddings routed to the same expert can be grouped and computed via a larger matrix multiplication. Expert-activation prediction and on-demand loading are not performed during prefilling due to the high number of expert activations in batched operations. Even if attempted, prediction offers minimal practical benefit, as all experts are likely to be invoked during batching³ and must be loaded in any case. Instead, we load the eight experts for each layer onto eight worker nodes in parallel (i.e., each worker handles one expert of every layer). Once loaded, embeddings are grouped by their desired experts, and batched expert activations are executed across the eight workers in parallel.

Our system splits a large batch into multiple mini-batches to improve GPU utilization on worker nodes. This design

³This claim is justified by our experimental results in Section 4. In the prefilling stage, short inputs with 16 tokens activates 7.6 experts out of total 8 experts on average, while long inputs with 128 tokens activate all 8 experts with 99.8% probability.

is motivated by a unique characteristic of edge scenarios: the LAN/WLAN link connecting the main node and worker nodes typically has lower throughput compared to the high-bandwidth, low-latency interconnect fabrics found in data centers. As a result, the communication cost for transmitting batched embeddings (from the main node to a worker) is non-negligible.

As illustrated in Fig. 7a, without mini-batching, the worker remains idle while the embedding vectors are being transmitted over the edge network, leading to reduced GPU utilization efficiency. In contrast, Fig. 7b shows that dividing a large batch into multiple mini-batches reduces the worker’s idle time by allowing it to begin expert computation as soon as the first mini-batch is received. In other words, mini-batching enables pipelined processing.

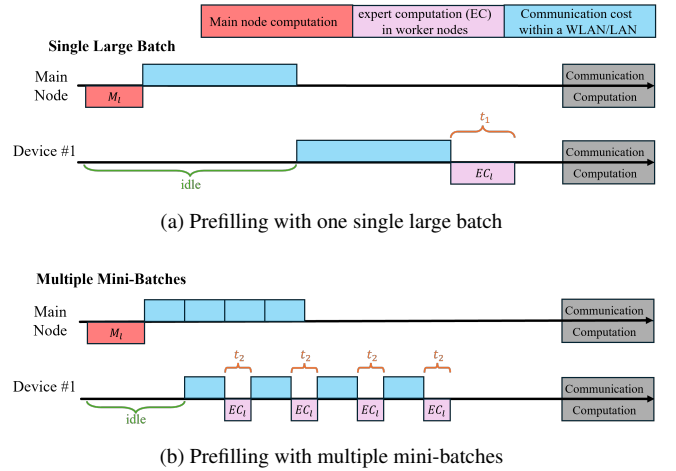


Figure 7: Illustrative timing diagrams for batched expert computations during the prefilling stage: (a) The main node transmits a single large batch to a worker. (b) The main node splits a large batch into mini-batches for transmission to a worker. We present the timing diagram for Device #1 only, since prefilling behaviors are similar across worker nodes.

Although the total computation time for the multiple mini-batches (i.e., $4t_2$ in Fig. 7b) is larger than that of a single large batch (i.e., t_1 in Fig. 7a) given the computation efficiency of a larger batch, mini-batched operation in Fig. 7b saves more time owing to the pipelined computation, eventually results in lower prefilling latency in general.

4 Experiments

This section begins with an overview of the experimental setup in Subsection 4.1. Subsection 4.2 investigates the alignment process through ablation studies and parameter optimizations. Subsection 4.3 compares the prediction accuracy of SEP with state-of-the-art baselines. Subsection 4.4 benchmarks OD-MoE against existing MoE inference frameworks.

4.1 Experimental Setup

Base Model and Configurations. We use Mixtral-8x7B [20] as the base model and employ the greedy decoding policy [29] for token generation. Greedy decoding selects the maximum-likelihood tokens in the output logits (i.e., not probabilistic sampling), ensuring better reproducibility.

Hardware Setup. Our testbed, as shown in Fig. 1, consists of ten nodes. The main node runs non-experts on a single NVIDIA RTX 3090 GPU, while the shadow node runs a quantized Mixtral-8x7B model (INT8, unless stated otherwise) on two NVIDIA RTX 3090 GPUs. The remaining eight worker nodes each have an NVIDIA RTX 3090 GPU. All nodes use AMD R7-7700 CPUs. Worker nodes have 192 GB of DRAM, while the main and shadow nodes each have 128 GB. The nodes are connected via a 1Gbps Ethernet LAN.

Speed Evaluation Metrics. Following the classic setup [24], we evaluate the inference speed of the decoding stage by its throughput, i.e., the average number of tokens generated per second during the decoding. The speed of the prefilling stage is measured by Time-To-First-Token (TTFT), and the overall speed is represented by the throughput for the whole inference process including the prefilling and decoding stages.

Test Setup (Inference Speed). Subsections 4.2 and 4.4 involve the evaluation of model inference speed. To facilitate reproduceable benchmarking across existing approaches, we follow the setup in previous works for speed test. Specifically, we inherit the test dataset from HOBBIT [31]. The test dataset contains a subset of 60 high-quality samples from the well-known Alpaca dataset [32], in which 30 of them have 16 input tokens while the remaining 30 samples have 128 input tokens.

All 60 prompts can generate at least 256 output tokens if left to run through their decoding iterations. However, we limit the number of iterations (output tokens) to either 64 or 256 in compiling our test results. The reason for doing so is to ensure fair comparisons between different schemes – ours as well as others – so that all schemes can be compared on the basis of the same number of decoding iterations (output tokens). Different inference frameworks may result in different output lengths if left to run, even with the same prompt, but all of them can also generate at least 256 output tokens.

The above setup results in four evaluation cases represented as (input length, output length) tuples: (16, 64), (16, 256), (128, 64), and (128, 256). For Subsection 4.2, we consider the (16, 256) configuration only, while we consider all the four input/output configurations in Subsection 4.4 for comprehensive benchmarking previous approaches.

4.2 Experimental Investigations of Alignment Process

This subsection investigates how SEP’s alignment setups impact OD-MoE’s decoding speed using the (16, 256) configuration. We conduct ablation studies to isolate the contributions

of KV cache alignment and token alignment to decoding speed. The six cases are as follows:

1. Shadow node enabled; token and KV cache alignments performed every iteration.
2. Shadow node enabled; only token alignment performed every iteration.
3. Shadow node enabled; only KV cache alignment performed every iteration.
4. Shadow node enabled; neither alignment operation performed.
5. Shadow node removed; worker nodes prefetch experts randomly.
6. Shadow node removed; worker nodes load experts only after receiving routing results from the main node.

To provide context regarding the data volume in the alignment process, we note that the full precision KV cache size on the main node is approximately 8 KB per token per decoding layer. In one alignment run, the main node sends all KV cache corresponding to the newly generated token to the shadow node, resulting in 256 KB (8 KB per token per layer times 32 decoding layers) of data transmission. The aligned token size is negligible (only a few bytes). The transmitted embedding size between the main node and a worker node is approximately 16 KB per token per layer. All data is transmitted over a 1 Gbps Ethernet connection.

Fig. 8 presents the results. The results of Cases 1–3 indicate that token alignment is more critical than KV-cache alignment – larger gap between Case 1 and 3 than the gap between Case 1 and 2. This is consistent with Fig. 3, where removing token alignment causes a larger drop in prediction accuracy.

On the other hand, the smaller yet noticeable gap between Cases 1 and 2 highlights that KV cache alignment is still essential for optimizing decoding speed. Despite the mean value in decoding speed, the increased standard deviation from Case 1 to Case 2 serves as another evidence in Fig. 8 that supports the necessity of KV cache alignment. Moreover, we note that experiments in Fig. 8 are conducted with 256 output tokens only – with more decoding iterations, removing the KV cache alignment as in Case 2 is expected to result in more severe deterioration in the decoding speed because the KV discrepancy accumulates as the number of token increase.

The drop in decoding speed from Case 4 to Case 5 also aligns with Fig. 3. In Case 4, the recall of the INT8-quantized model without KV or token alignment (i.e., the blue curve in Fig. 3b) decreases from 100% to around 30% as the token index increases from 1 to 256, yielding an average recall of ~45%, whereas an easy calculation shows that the recall in Case 5 is only ~25%. The higher recall in Case 4 explains its better decoding speed in comparison to Case 5.

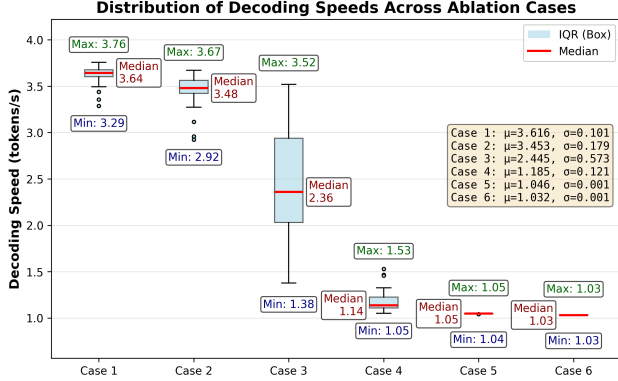


Figure 8: Average decoding speed (tokens/s) for different ablation setups.

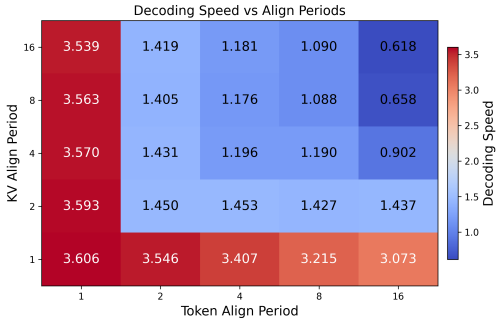


Figure 9: Average decoding speed (tokens/s) of OD-MoE under different token and KV-cache alignment periods.

Finally, the small additional drop in decoding speed from Case 5 to Case 6 suggests that random preloading offers limited benefit. As a general takeaway, the monotonic decrease from Case 1 to Case 6 demonstrates the effectiveness of each component in our MoE inference framework.

Following the ablation study, we next study the optimal alignment periods for tokens and KV cache. Recall from Section 3.2 that alignment introduces a trade-off between late-departure latency and reduced prediction error. Fig. 9 reports the decoding speeds for the alignment periods of 1/2/4/8/16 for both token and KV-cache. As shown, the best decoding speed occurs when both token and KV-cache alignments use a period of one. In subsequent experiments comparing OD-MoE with baselines (see Subsection 4.4), we adopt this configuration.

We emphasize that although the highest speed in Fig. 8 is achieved with a simple configuration (i.e., aligning tokens and KV cache every decoding iteration), this does not mean the trade-off posed by the late-departure issue discussed in Section 3.3 is not important in general. The results in Fig. 9 only indicate that reducing prediction error has a much larger impact on speed than lowering late-departure latency with the experimental hardware setup. In general, the optimal trade-off

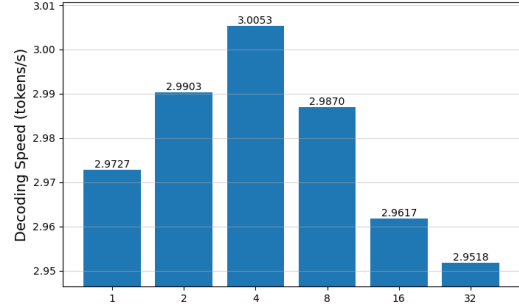


Figure 10: Average decoding speed (tokens/s) of OD-MoE with worker-node GPUs replaced by RTX 3080s. The token-alignment period is fixed at 1; the KV-cache alignment period varies over {1, 2, 4, 8, 16, 32}.

depends on both computation time (expert activation) and communication time (expert loading) on the worker node. For example, replacing the worker-node GPUs with RTX 3080s (see Fig. 10) shifts the optimum: the best speed is obtained with a KV-cache alignment period of four and a token-alignment period of one.

4.3 Baseline Comparison: Expert-Activation Prediction

Table 1 lists the performance results of some previously reported expert-activation prediction schemes. Prediction accuracy can be evaluated using two metrics: recall rate and cache-hit rate (the probability of having the required expert in the GPU cache).

Table 1: Baseline Comparison in expert-activation prediction.

	Adap	DAOP	Hobt	MxOf	fMoE
Recall Rate	0.86	0.84	0.91	/	/
Cache-hit Rate	/	/	/	~0.80	<0.85

Note: Adap = AdapMoE, Hobt = HOBBIT, MxOf = Mixtral Offloading

AdapMoE [42] and DAOP [41] achieve recall rates of 86% and 84%, respectively. HOBBIT [31] is more sophisticated – it allows predictions across multiple layers. Specifically, HOBBIT makes predictions up to four layers ahead, and it reports a recall rate of 91% on average for predicted layers.

Mixtral Offloading [15] and fMoE [38] evaluate their predictor by cache-hit rate. As these two projects are either partly open source or closed source, we cannot reproduce their prediction algorithms. However, based on figures in their technical reports, we can roughly estimate their cache-hit rate: the Fig. 2 in [15] indicates that its cache-hit rate is around 80% when applying the same top-2 activation policy as in this

paper, while the Fig. 12 in [38] shows that fMoE’s cache-hit rate is no larger than 85%.

As OD-MoE follows the cache-free design (i.e., only the predicted expert is loaded into the GPU), the cache-hit rate is equivalent to its recall rate. With the test setup described in Section 3.2, experimental results presented in Fig. 3 highlight that SEP achieves 99.94%, 97.34%, and 95.67% recall rates when realized with FP16, INT8, and NF4 quantization, respectively. These experimental results, even with the NF4 quantization, clearly outperform existing methods in both prediction recall rate and cache-hit rate⁴.

4.4 Baseline Comparison: Inference Performance

This subsection benchmarks OD-MoE against existing methods based on three metrics:

1. **Model Inference Speed:** This includes the speeds during the prefilling stage, the decoding stage, and the overall inference process.
2. **GPU Memory:** The total GPU memory required by the inference system.
3. **Model Performance:** Evaluation on multiple classic LLM benchmarks to measure the quality of the model’s answers.

We compare OD-MoE against two classes of baselines: 1) full-precision inference engines with all experts pre-cached, and 2) representative expert-offloading systems. The former, which includes HuggingFace Transformers [33] and llama.cpp [16], serves as data-center references where the MoE model under study (i.e., Mixtral-8x7B) does not require expert offloading. The latter includes Mixtral-Offloading [15], MoE-Infinity [34], HOBBIT [31], and AdapMoE [42] – existing open-source systems discussed previously.

To deploy the full-precision Mixtral-8x7B through the Transformers engine, GPU memory of at least equivalent to eight RTX 3090 GPUs is required. Therefore we reproduce the baseline methods on a GPU server with 1) eight NVIDIA RTX 3090 GPUs, 2) AMD EPYC 7K62 CPU, and 3) 512G DRAM. Note that the offloading baselines do not support distributed computing and thus only leverage one GPU on the server. This hardware setup ensures faithful reproductions of baseline methods and fair comparisons with our system.

Our experimental results are summarized in Table 2. We first look at Part (i) of Table 2. For *decoding throughput* – the primary focus of this paper – the Transformers engine with all experts cached achieves 4.8900 tokens/s on average (averaged

across four input/output configurations). Note that we do not consider batched decoding here to align with the decoding setup of previous studies [15, 31, 34, 42] for fair comparisons. OD-MoE achieves 3.6925 tokens/s on average, retaining 75.51% of the Transformers engine’s speed. Compared with llama.cpp (0.8225 tokens/s on average), which runs on CPUs and caches all expert parameters in DRAM, OD-MoE is 4.49× faster owing to GPU acceleration. Against expert-offloading baselines, OD-MoE outperforms Mixtral-Offloading (2.2375 tokens/s), MoE Infinity (0.6875 tokens/s), HOBBIT (0.7850 tokens/s), and AdapMoE (3.1300 tokens/s) by 1.65×, 5.37×, 4.70×, and 1.18×, respectively.

We next examine *prefilling speed*, measured by TTFT. Prefilling latency is strongly dependent on input length: OD-MoE achieves ~1340 ms and ~3150 ms TTFT for 16 and 128 input tokens, respectively, with an average of ~2244 ms. Compared with offloading baselines, OD-MoE outperforms MoE Infinity and HOBBIT but trails Mixtral-Offloading and AdapMoE – likely because these two systems use expert quantization to reduce both loading and computation time. However, as shown in Part (iii) of Table 2, expert quantization degrades answer quality.

For the output throughput encompassing both prefilling and decoding stages, OD-MoE averages 3.3700 tokens/s, exceeding Mixtral-Offloading (2.1725 tokens/s), MoE Infinity (0.6675 tokens/s), HOBBIT (0.7575 tokens/s), and AdapMoE (3.0350 tokens/s) by 1.55×, 5.05×, 4.45×, and 1.11×, respectively.

Moving to Part (ii) of Table 2 on **GPU memory requirement**, OD-MoE uses only 1/3 of the full-precision model’s GPU memory. A detailed breakdown is as follows: 7 GB on the main node, 45 GB on the shadow node, and 1 GB per worker across eight workers, resulting in 60GB in total. As llama.cpp targets CPU-oriented computations, its GPU memory usage is nil.

Finally, we look at Part (iii) of Table 2, answer quality. The evaluation of an LLM’s performance is a broad topic. To narrow down the scope, we select widely used LLM performance benchmarks across six representative categories of interest to the community: 1) General Knowledge [19, 39], 2) Math [10, 11], 3) Reasoning [27, 30], 4) Coding [7, 43], 5) Instruction Following [3], and 6) Anti-Hallucination [22]. All these benchmarks have open-sourced the evaluation scripts, and most of them present the evaluation results with the percentage of correct/satisfactory answers. The only exception is MT-bench-101 [3], which uses a third-party judge (i.e., following the setup in [3], we use GPT4 as the judge) to score the tested model’s instruction following performance under the setup of multi-turn dialogues. As a full-precision solution, OD-MoE matches the answer quality of Transformers and llama.cpp. Meanwhile, as experimental results suggest, OD-MoE consistently surpasses expert-offloading baselines across all benchmarks tested within the six categories.

To conclude the discussion of the results, we remark that

⁴Aside from the works discussed here, we do not benchmark against other works mentioned in Section 2, as they are either partially open source and do not provide the predictor implementation, or they require specialized mobile devices for system deployment. Moreover, their technical reports do not provide direct information on prediction accuracies or cache-hit rates.

Table 2: Baseline Comparison of inference speed, GPU memory requirement, and model answer quality.

Category	Configuration/Dataset	Mixtral Offloading	MoE-Infinity	HOBBIT	AdapMoE	Transformer	Llama.cpp	OD-MoE
(i) Benchmarks in Inference Speed								
TTFT (ms)	(16, 64)	1727.81	5521.63	5456.37	1345.12	385.52	2025.10	1349.78
	(16, 256)	1705.15	5560.34	5467.42	1428.23	386.86	2008.60	1340.89
	(128, 64)	1967.16	5819.49	5898.92	1343.96	447.43	6592.81	3150.87
	(128, 256)	1979.81	7573.27	5896.04	1430.69	448.16	6527.13	3135.85
	Average	1844.9825	6118.6825	5679.6875	1387.0000	416.9925	4288.4100	2244.3475
Decoding Throughput (token/s)	(16, 64)	2.26	0.68	0.78	3.14	4.90	0.85	3.67
	(16, 256)	2.26	0.68	0.79	3.15	4.87	0.82	3.65
	(128, 64)	2.22	0.70	0.79	3.11	4.90	0.81	3.74
	(128, 256)	2.21	0.69	0.78	3.12	4.89	0.81	3.71
	Average	2.2375	0.6875	0.7850	3.1300	4.8900	0.8225	3.6925
Output Throughput (token/s)	(16, 64)	2.16	0.66	0.74	2.99	4.84	0.84	3.41
	(16, 256)	2.24	0.67	0.78	2.98	4.86	0.82	3.54
	(128, 64)	2.11	0.66	0.74	3.09	4.81	0.75	3.09
	(128, 256)	2.18	0.68	0.77	3.08	4.86	0.78	3.44
	Average	2.1725	0.6675	0.7575	3.0350	4.8425	0.7975	3.3700
(ii) Benchmarks in total GPU Memory Requirement (GB)								
Baseline Models Implemented by Default		11	21.5	22	8	180	N/A	60
(iii) Benchmarks in Answer Quality								
General Knowledge	Hellaswag	63.50%	64.50%	62.00%	44.50%		76.25%	
	MMLU	49.82%	51.93%	55.26%	48.60%		70.34%	
Math	ARC-Challenging	82.37%	77.97%	86.10%	75.68%		86.44%	
	GSM8k	56.00%	60.00%	35.00%	22.00%		64.14%	
Reasoning	WinoGrande	52.40%	56.40%	41.20%	43.20%		65.67%	
	BigBenchHard	58.30%	52.20%	38.89%	43.33%		59.84%	
Coding	Big Code	8.00%	15.00%	8.00%	0.00%		16.00%	
	humaneval	17.68%	1.83%	1.83%	1.54%		24.39%	
Instruction Following	MT-bench-101	7.4 / 10	7.81 / 10	6.67 / 10	4.47 / 10		7.83 / 10	
Anti-Hallucination	truthfulQA	86.50%	87.50%	73.50%	76.50%		89.00%	

OD-MoE, being a full-precision MoE distributed inference framework with expert offloading, preserves full-precision answer quality while using comparatively little GPU memory compared with the fully pre-cached solution. OD-MoE demonstrates consistent answer quality advantages over all the offloading baselines which have precision losses, and exhibits superior inference speed comparable to the fully-precached solution and surpassing all offloading baselines. However, the full-precision and competitive inference speed comes with a price of a large GPU memory usage for the shadow node and the network delays absent in the baselines' single-server systems. On the other hand, OD-MoE has the advantage of parallel execution over distributed nodes. More importantly, the worker nodes only require a small GPU memory footprint. The RTX 3090 GPUs in the testbed can be substituted with less powerful entry-level GPUs.

5 Conclusion

This paper presented **OD-MoE**, a cache-free, on-demand MoE inference framework designed specifically for

memory-constrained edge environments. The core innovations of OD-MoE include: **1) SEP**, an ultra-accurate, multi-layer lookahead predictor implemented with a quantized shadow model, reaching up to 99.94% accuracy for expert-activation predictions; and **2) Grouped workers with round-robin scheduling**, which overlap expert loading and computation across distributed devices. This parallelization significantly increases effective CPU-GPU I/O throughput and eliminates the need for long-term expert caching.

To ensure stability during long decoding sequences, we introduced an advanced alignment mechanism in SEP, which synchronizes the shadow model's token generation and KV-cache with the full-precision model. On Mixtral-8 $\times$ 7B, OD-MoE achieves 75% of the decoding speed of a full GPU-cached deployment while using only one-third of the GPU memory. The framework maintains full-precision answer quality and consistently outperforms representative expert-offloading baselines across multiple evaluation benchmarks. Additionally, OD-MoE reduces the per-worker GPU memory footprint to under 1 GB, enabling practical MoE inference on low-cost edge GPUs and even IoT-class devices.

References

- [1] Beating google and apple, huawei brings large ai model to mobile voice assistant. <https://www.huaweicentral.com/beating-google-and-apple-huawei-brings-large-ai-model-to-mobile-voice-assistant/>, 2023. Accessed: 2025-11-20.
- [2] APPLE. Introducing apple’s on-device and server foundation models. <https://machinelearning.apple.com/research/introducing-apple-foundation-models>, 2024. Accessed: 2025-11-20.
- [3] BAI, G., LIU, J., BU, X., HE, Y., LIU, J., ZHOU, Z., LIN, Z., SU, W., GE, T., ZHENG, B., ET AL. Mt-bench-101: A fine-grained benchmark for evaluating large language models in multi-turn dialogues. *arXiv preprint arXiv:2402.14762* (2024).
- [4] BAI, Y., ZHANG, J., LV, X., ZHENG, L., ZHU, S., HOU, L., DONG, Y., TANG, J., AND LI, J. Longwriter: Unleashing 10,000+ word generation from long context llms. *arXiv preprint arXiv:2408.07055* (2024).
- [5] CAO, S., LIU, S., GRIGGS, T., SCHAFHALTER, P., LIU, X., SHENG, Y., GONZALEZ, J. E., ZAHARIA, M., AND STOICA, I. 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* (2025), pp. 715–730.
- [6] CHEN, K., DU, Y., LI, J., CAO, H., GUO, M., DANG, X., LI, L., QIU, J., CHEN, G., AND HENG, P. A. Chemminer: A large language model agent system for chemical literature data mining. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2025), pp. 7595–7603.
- [7] CHEN, M. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).
- [8] CHEN, Q., CHEN, X., AND HUANG, K. Slimcaching: Edge caching of mixture-of-experts for distributed inference. *arXiv preprint arXiv:2507.06567* (2025).
- [9] CHU, X., QIAO, L., ZHANG, X., XU, S., WEI, F., YANG, Y., SUN, X., HU, Y., LIN, X., ZHANG, B., ET AL. Mobilevlm v2: Faster and stronger baseline for vision language model. *arXiv preprint arXiv:2402.03766* (2024).
- [10] CLARK, P., COWHEY, I., ETZIONI, O., KHOT, T., SABHARWAL, A., SCHOENICK, C., AND TAFJORD, O. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457* (2018).
- [11] COBBE, K., KOSARAJU, V., BAVARIAN, M., CHEN, M., JUN, H., KAISER, L., PLAPPERT, M., TWOREK, J., HILTON, J., NAKANO, R., ET AL. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168* (2021).
- [12] COIN POET. GPUs Ranked by \$ / Memory Bandwidth (GB/s). <https://coinpoet.com/ml/learn/gpu/ranking/memory-bandwidth-gbs>, 2023. Accessed: 2025-11-20.
- [13] DANG, X., CHEN, K., SU, X., NOORI, A., ARANGO, I., VITTOR, L., LONG, X., DU, Y., ZITNIK, M., AND HENG, P. A. Knowguard: Knowledge-driven abstention for multi-round clinical reasoning. *arXiv preprint arXiv:2509.24816* (2025).
- [14] DU, Y., YANG, Q., WANG, L., LIN, J., CUI, H., AND LIEW, S. C. Llmind 2.0: Distributed iot automation with natural language m2m communication and lightweight llm agents. *arXiv preprint arXiv:2508.13920* (2025).
- [15] ELISEEV, A., AND MAZUR, D. Fast inference of mixture-of-experts language models with offloading. *arXiv preprint arXiv:2312.17238* (2023).
- [16] GERGANOV, G. ggerganov/llama.cpp: Port of facebook’s llama model in c/c++. <https://github.com/ggml-org/llama.cpp>, 2023.
- [17] GHOLAMI, A., KIM, S., DONG, Z., YAO, Z., MAHONEY, M. W., AND KEUTZER, K. A survey of quantization methods for efficient neural network inference. In *Low-power computer vision*. Chapman and Hall/CRC, 2022, pp. 291–326.
- [18] HAN, Y., PAN, L., PENG, J., TAO, Z., ZHANG, W., AND ZHANG, Y. Grace-moe: Grouping and replication with locality-aware routing for efficient distributed moe inference. *arXiv preprint arXiv:2509.25041* (2025).
- [19] HENDRYCKS, D., BURNS, C., BASART, S., ZOU, A., MAZEIKA, M., SONG, D., AND STEINHARDT, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300* (2020).
- [20] 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. Mixtral of experts. *arXiv preprint arXiv:2401.04088* (2024).
- [21] LI, N., GUO, S., ZHANG, T., LI, M., HONG, Z., ZHOU, Q., YUAN, X., AND ZHANG, H. The moe-empowered edge llms deployment: Architecture, challenges, and opportunities. *arXiv preprint arXiv:2502.08381* (2025).
- [22] LIN, S., HILTON, J., AND EVANS, O. Truthfulqa: Measuring how models mimic human falsehoods. In *Proceedings of the 60th annual meeting of the association for computational linguistics (volume 1: long papers)* (2022), pp. 3214–3252.
- [23] LIU, J., TANG, P., WANG, W., REN, Y., HOU, X., HENG, P.-A., GUO, M., AND LI, C. A survey on inference optimization techniques for mixture of experts models. *arXiv preprint arXiv:2412.14219* (2024).
- [24] MITZENMACHER, M., AND SHAHOUT, R. Queueing, predictions, and large language models: Challenges and open problems. *Stochastic Systems* 15, 3 (2025), 195–219.
- [25] POLINO, A., PASCANU, R., AND ALISTARH, D. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).
- [26] QUALCOMM TECHNOLOGIES, INC. World’s first on-device demonstration of Stable Diffusion on Android. <https://www.qualcomm.com/news/onq/2023/02/worlds-first-on-device-demonstration-of-stable-diffusion-on-android>, 2023. Accessed: 2025-11-20.
- [27] SAKAGUCHI, K., BRAS, R. L., BHAGAVATULA, C., AND CHOI, Y. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM* 64, 9 (2021), 99–106.
- [28] SHAHOUT, R., CAI, C., DU, Y., YU, M., AND MITZENMACHER, M. From score distributions to balance: Plug-and-play mixture-of-experts routing. *arXiv preprint arXiv:2510.03293* (2025).
- [29] SONG, Y., WANG, G., LI, S., AND LIN, B. Y. The good, the bad, and the greedy: Evaluation of llms should not ignore non-determinism. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (2025), pp. 4195–4206.
- [30] SUZGUN, M., SCALES, N., SCHÄRLI, N., GEHRMANN, S., TAY, Y., CHUNG, H. W., CHOWDHURY, A., LE, Q., CHI, E., ZHOU, D., AND WEI, J. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023* (Toronto, Canada, July 2023), A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds., Association for Computational Linguistics, pp. 13003–13051.
- [31] TANG, P., LIU, J., HOU, X., PU, Y., WANG, J., HENG, P.-A., LI, C., AND GUO, M. Hobbit: A mixed precision expert offloading system for fast moe inference. *arXiv preprint arXiv:2411.01433* (2024).
- [32] TAORI, R., GULRAJANI, I., ZHANG, T., DUBOIS, Y., LI, X., GUESTRIN, C., LIANG, P., AND HASHIMOTO, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.
- [33] WOLF, T., DEBUT, L., SANH, V., CHAUMOND, J., DELANGUE, C., MOI, A., CISTAC, P., RAULT, T., LOUF, R., FUNTOWICZ, M., ET AL. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (2020), pp. 38–45.

- [34] XUE, L., FU, Y., LU, Z., MAI, L., AND MARINA, M. Moe-infinity: Offloading-efficient moe model serving. *arXiv preprint arXiv:2401.14361* (2024).
- [35] XUE, N., SUN, Y., CHEN, Z., TAO, M., XU, X., QIAN, L., CUI, S., AND ZHANG, P. Wdmoe: Wireless distributed large language models with mixture of experts. In *GLOBECOM 2024-2024 IEEE Global Communications Conference* (2024), IEEE, pp. 2707–2712.
- [36] XUE, Z., SONG, Y., MI, Z., ZHENG, X., XIA, Y., AND CHEN, H. Powerinfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282* (2024).
- [37] YI, R., GUO, L., WEI, S., ZHOU, A., WANG, S., AND XU, M. Edge-moe: Fast on-device inference of moe-based large language models. *arXiv preprint arXiv:2308.14352* (2023).
- [38] YU, H., CUI, X., ZHANG, H., AND WANG, H. fmoe: Fine-grained expert offloading for large mixture-of-experts serving. *arXiv preprint arXiv:2502.05370* (2025).
- [39] ZELLERS, R., HOLTZMAN, A., BISK, Y., FARHADI, A., AND CHOI, Y. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830* (2019).
- [40] ZHANG, P., ZENG, G., WANG, T., AND LU, W. Tinyllama: An open-source small language model. *arXiv preprint arXiv:2401.02385* (2024).
- [41] ZHANG, Y., AGGARWAL, S., AND MITRA, T. Daop: Data-aware offloading and predictive pre-calculation for efficient moe inference. In *2025 Design, Automation & Test in Europe Conference (DATE)* (2025), IEEE, pp. 1–7.
- [42] ZHONG, S., LIANG, L., WANG, Y., WANG, R., HUANG, R., AND LI, M. 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* (2024), pp. 1–9.
- [43] ZHUO, T. Y., VU, M. C., CHIM, J., HU, H., YU, W., WIDYASARI, R., YUSUF, I. N. B., ZHAN, H., HE, J., PAUL, I., ET AL. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. *arXiv preprint arXiv:2406.15877* (2024).