Efficient MoE Inference on Single GPU with Dynamic Expert Caching

Rui Zhang, Boxuan Yang, Rongji Wang, Xuemei Peng, Zeyi Wen*
The Hong Kong University of Science and Technology (Guangzhou)
Guangzhou, China
Emails: {rui.zhang, rwang135, xpeng558}@connect.hkust-gz.edu.cn, {boxuanyang, wenzeyi}@hkust-gz.edu.cn

***Abstract*—The Mixture-of-Experts (MoE) architecture has demonstrated notable advantages by enabling model scaling without a proportional increase in computational cost. However, due to the large model size, deployment typically requires substantial memory resources that are often unavailable on consumer-grade devices. MoE offloading strategies have been proposed to address this. However, PCIe bandwidth constraints lead to noticeably slower inference during offloading. Several methods, such as expert caching, prefetching, and hybrid CPU/GPU inference, have been introduced to mitigate this issue. Yet, they still suffer from load imbalance across devices, low hardware utilization, and significant transfer latency, which become particularly severe when handling ultra-large models (e.g., $\geq$ 400B). To address these challenges, this paper proposes SMOE, a hybrid CPU/GPU inference framework for MoE LLMs based on dynamic expert caching, to improve inference efficiency and hardware utilization on consumer-grade devices. SMOE adopts specialized optimization strategies for the prefilling and decoding stages. For the prefilling, it employs an on-demand expert loading strategy optimized with pipeline techniques to schedule experts at runtime dynamically. For the decoding, it introduces a novel token-wise prefetch strategy that maximizes computation-communication overlap, enabling expert scheduling across CPU and GPU without incurring noticeable transfer delays. In addition, SMOE implements an efficient cache management strategy to make use of the limited GPU memory. Experimental evaluations on consumer-grade devices with ultra-large MoE models demonstrate that SMOE achieves a speedup of up to 8.68$\times$ over the baseline and 2.98$\times$ over the SOTA method in the prefilling stage, with the benefit increasing as the length grows. In the decoding stage, SMOE achieves up to 20.9% faster inference compared to the baseline, while other caching- and prefetching-based methods fail to handle ultra-large models such as 671B. These results show that SMOE achieves efficient inference acceleration for ultra-large MoE models on consumer-grade devices.**

***Index Terms*—Large Language Model, MoE Inference, Offloading.**

I. Introduction

Large Language Models (LLMs) have achieved remarkable progress and have been widely applied in areas such as NLP [1]–[3], code generation [4], [5], and AI agents [6], [7]. While their substantial computational requirements generally confine deployment on high-end servers, there is a growing need to deploy LLMs on consumer-grade local platforms, driven by concerns over data privacy as well as requirements for model customization [8]. Offloading is a common method for deploying LLMs on consumer-grade devices. Typical methods include layer-wise offloading, such as Llama.cpp [9], and neuron-level offloading, such as PowerInfer [8].

The emergence of Mixture-of-Experts (MoE) has opened new opportunities for deploying increasingly larger LLMs on consumer-grade devices. By employing a gating function to activate a subset of experts selectively, MoE significantly reduces computational costs without substantially compromising model performance [10], [11]. Due to its reduced computation and the fact that not all weights are involved, many offloading strategies [12], [13] have been proposed to enable expert offloading deployment. These approaches leverage secondary storage, such as CPU memory or SSDs, to hold model weights and dynamically load them into HBM at runtime. While offloading strategies alleviate memory bottlenecks, they remain constrained by the limited PCIe bandwidth, particularly in consumer-grade platforms, which often results in substantial data transfer latency and degraded inference efficiency.

To address this issue, recent studies have explored techniques such as quantization [14], prefetching [15], [16], caching [14], and CPU/GPU hybrid computation [12], [13] to either hide or mitigate data transfer latency. These methods can be broadly categorized into two types based on whether CPU computation is involved. The first class is GPU-centric methods, which primarily rely on GPU computation while using the large memory capacity of the DRAM merely for expert storage [14]–[22]. During inference, only the required experts are transferred from the CPU to the GPU. Such methods either introduce significant data transfer latency, or approximate the computation by skipping experts that are not loaded onto the GPU in time, resulting in degraded model performance. The second class is CPU/GPU hybrid inference methods, which utilize the CPU to handle cases of expert cache misses on the GPU [13], [23]. This approach reduces data transfer overhead while maintaining computational accuracy. However, a major challenge of this category lies in the load imbalance between the CPU and GPU, which makes it challenging to achieve optimal acceleration. Typically, these methods estimate expert activation frequencies based on historical statistics, classify experts as hot or cold, and place them on the GPU or CPU accordingly. At runtime, prefetching mechanisms are used for dynamic expert scheduling to achieve better load balancing.

These approaches perform well for small- and medium-scale

*Corresponding author.

models (e.g., $\leq$ 100B) [24], where the GPU can cache a large proportion of experts. However, for large-scale models, the limited memory of consumer-grade GPUs can only cache a small subset of experts, leading to a low cache hit rate. Moreover, in modern MoE-based LLMs, expert activation patterns tend to be increasingly uniform, reducing the effectiveness of caching only hot experts. Consequently, prefetching techniques become essential for timely cache updates to improve the hit rate. Existing prefetching methods can be roughly divided into iteration-wise and layer-wise approaches. Iteration-wise methods predict the expert activation patterns for an entire iteration in advance to guide prefetching. While shallow-layer predictions tend to be accurate, they leave insufficient time for data transfer, whereas deeper-layer predictions are less accurate and less effective. In contrast, layer-wise methods guide prefetching through cross-layer prediction, achieving high accuracy but lacking sufficient computation time to fully overlap the data transfer latency.

To improve hardware utilization and enable more efficient MoE inference on consumer-grade GPUs, we propose SMOE, a hybrid MoE inference framework that applies distinct optimization strategies for the prefilling and decoding stages based on dynamic expert caching. For the prefilling stage, which is computation-intensive, SMOE employs an on-demand loading strategy that dispatches computationally heavy experts to the GPU, while lighter experts are executed on the CPU, because the acceleration benefit of executing compute-intensive experts on the GPU far outweighs the overhead introduced by expert transfers. Moreover, SMOE employs a pipeline mechanism to further hide the transfer latency.

For the decoding stage which involves lighter computation and the overhead of on-demand loading becomes non-negligible, SMOE adopts a caching and prefetching strategy for dynamic expert scheduling. Specifically, based on our empirical observations of expert activation behavior, we introduce a novel token-wise prefetching method, which predicts the experts likely to be activated by the next token at each MoE layer and prefetches them in advance. Compared with existing prefetching methods, our method improves the number of experts transferred over PCIe within a limited time budget, thereby improving the expert cache hit rate and achieving a more balanced workload between the CPU and GPU, which in turn accelerates decoding. By transferring more experts, the GPU performs more computation, leading to higher PCIe and GPU utilization. SMOE also offers greater flexibility by dynamically selecting specific MoE layers for prefetching, thereby reallocating the time saved from inaccurately predicted layers to those with higher prediction accuracy, which in turn improves the expert cache hit rate. In addition, SMOE incorporates an efficient cache management mechanism that enables effective utilization of limited GPU memory when scaling to extremely large models.

Experimental results on consumer-grade GPUs with the cutting-edge MoE LLMs, DeepSeek-V3-671B and Qwen3-235B, under single-concurrency evaluation demonstrate the effectiveness of SMOE. In the prefilling stage, when the prompt length is 400, SMOE achieves approximately 2.64$\times$ speedup over the baseline, and the improvement grows with longer prompts, reaching 8.68$\times$ at a length of 4K. Compared to the SOTA method, SMOE achieves up to 2.98$\times$ speedup. In the decoding stage, SMOE delivers up to 20.9% speedup compared with the baseline, while other caching- and prefetching-based methods fail to handle large models. The advantage arises because, with extremely large models, the limited memory of the GPU can only cache a small proportion of experts. Moreover, traditional prefetching lacks flexibility and fails to fully achieve computation-communication overlapping, preventing timely expert cache updates and thus limiting its effectiveness. In summary, the key contributions in this paper are as follows:

- We propose a CPU/GPU hybrid inference framework for MoE models, featuring distinct optimization strategies tailored to the characteristics of the prefilling and decoding stages.
- We introduce a novel token-wise prefetching method that achieves better computation-communication overlapping and flexibility, thereby improving cache hit rate and bandwidth utilization compared with traditional approaches.
- To address the limitation of expert cache capacity when dealing with large-scale models, we design an efficient cache management scheme that improves cache hit rates and alleviates CPU/GPU load imbalance.
- Experimental result shows that our framework demonstrates effective MoE offloading and improved inference efficiency on ultra-large models such as DeepSeek-V3 671B, while other state-of-the-art methods yield negligible or even adverse speedup effects.

II. Background

A. Mixture of Experts

Scaling laws demonstrate that the large language model's performance improved predictably with increases in the model size, dataset size, and training compute [25]. However, the large model size has led to enormous computational demands. Mixture of Experts (MoE) architecture was introduced to scale up the model size while avoiding excessive growth in computational cost [11]. It replaces the dense layers with multiple smaller experts, and a gating function is employed to selectively activate a subset of experts, thereby reducing the computational cost. The implementation of MoE varies across large language models. For instance, Mixtral adopts an MoE module with eight experts, of which two are activated for each token [26]. DeepSeek employs larger expert pools, with the V2 version containing 64 experts and activating six per token [27], and the V3 version scaling to 256 experts with eight activated per token [28]. Furthermore, DeepSeek introduces shared experts to enhance representation capacity and model expressiveness.

B. MoE Offloading

Weight offloading strategies have been proposed to address the memory constraints of large language models, which are

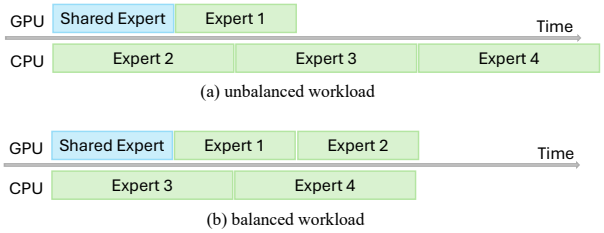


Fig. 1: Balanced and Unbalanced CPU/GPU Execution.

particularly pronounced on consumer-grade devices. These methods leverage CPU memory as secondary storage when GPU memory is insufficient, dynamically loading weights into the GPU on demand. Owing to their sparse activation property, MoE models are especially well-suited for offloading. For example, in DeepSeek-V3, MoE modules account for 98% of the model's weights, whereas the remaining modules (e.g., attention layers) account for only 2% approximately. Each MoE layer has 256 experts, and eight are activated per token.

Existing MoE offloading strategies can be broadly categorized into GPU-centric and CPU–GPU hybrid approaches. GPU-centric approaches often face a trade-off between model accuracy and inference speed. To preserve model fidelity, experts must be loaded on-demand from CPU memory, inevitably introducing transfer latency (e.g., [22]). Conversely, methods aiming for faster inference often skip experts not resident in GPU memory, as in AdapMoE [21] and SIDA-MoE [18]. CPU/GPU hybrid approaches mitigate model degradation by computing some experts directly on the CPU. For instance, KTransformers [12] assigns all routed experts to the CPU and the remaining computations to the GPU, thus avoiding loading the weight-heavy MoE layers onto the GPU. However, this serial inter-device execution forces the GPU to idle while waiting for CPU computations, leading to poor utilization. To overcome this, recent works such as HybriMoE [13] and Fiddler [23] propose a hybrid execution of experts across CPU and GPU. These methods rely on expert caching and prediction mechanisms, including training-based, statistical, and pro-gating approaches, to prefetch potentially activated experts onto the appropriate device. In theory, such approaches can achieve optimal speedup when CPU–GPU workloads are balanced, as illustrated in Figure 1. However, load balancing is highly dependent on cache hit rate. For small- to medium-scale models, consumer GPUs can cache a relatively large number of experts, and the skewed expert activation patterns of many models yield high hit rates. In contrast, for extremely large models such as DeepSeek-V3 671B, the limited memory capacity of consumer GPUs constrains the number of cached experts. Without an effective prefetching strategy to refresh the cache in a timely manner, acceleration efficiency degrades significantly.

Prefetching strategies can generally be classified into iteration-wise and layer-wise methods as shown in Figure 2. Iteration-wise prefetching typically relies on predictors or historical statistics to load experts before an entire forward

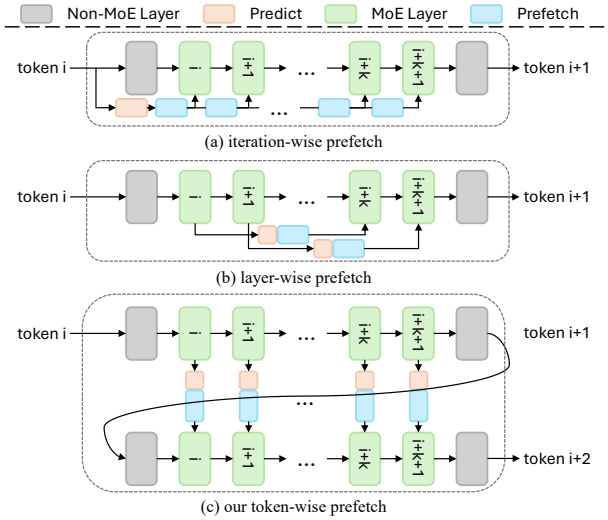


Fig. 2: Layer-wise, Iteration-wise, and Our Token-wise Prefetch Methods Comparison.

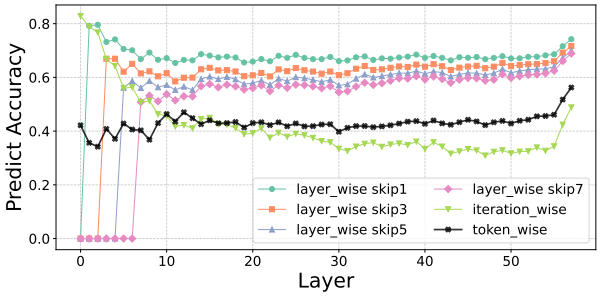


Fig. 3: Prediction Accuracy for Layer-wise, Iteration-wise, and Our Token-wise Methods on DeepSeek-V3 Model.

pass, as illustrated in Figure 2 (a). However, the prediction accuracy of its deeper layers is limited, as shown in Figure 3, and for the high-accuracy layers, there isn't enough time for data transfer. In contrast, layer-wise prefetching exploits the similarity of activation values across layers, caused by residual connections, to achieve higher prediction accuracy. Yet, the small computation window at each layer leaves insufficient time for prefetching. Some works propose cross-layer or sliding-window prefetching to extend the available time. Nevertheless, our experimental analysis indicates that such mechanisms often cause the accumulation of transfer tasks, as shown in Figure 2, failing to deliver improvements.

III. Motivation

A. Insight 1: Distinct Characteristics in Prefill and Decode

As illustrated in Figure 4, the prefilling and decoding stages exhibit fundamentally different characteristics, which necessitate stage-specific optimization strategies. The prefilling stage processes all input tokens within a single forward pass, making it highly computational-intensive. Its computational cost grows rapidly with the prompt length, while device memory typically remains limited and can only hold a fraction of the

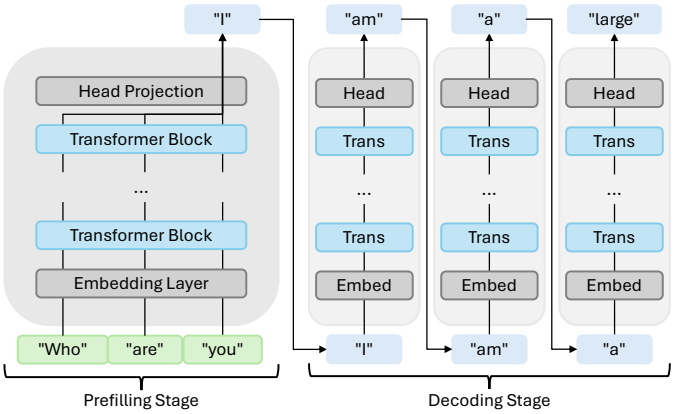
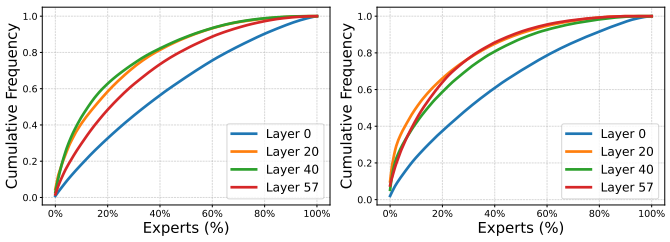


Fig. 4: Prefilling and Decoding Stages in LLMs Inference.



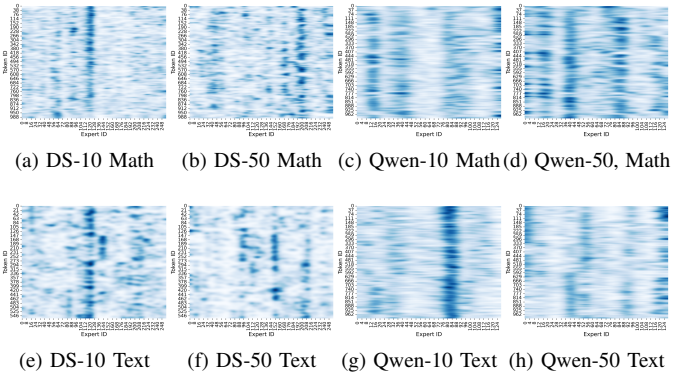
(a) Math (b) Text Summarization

Fig. 5: Expert Activation Cumulative Frequency in Different Layers on Different Tasks. Where deeper layers show more activation skewness than shallow layers.

experts. Since the transfer time for loading experts is nearly constant, once the input length exceeds a certain threshold, the benefit of transferring heavily utilized experts to the GPU outweighs the associated overhead. In contrast, the decoding stage is memory-intensive. Although its computational demand is smaller than that of the prefilling stage, it requires more frequent access to different experts. In this case, efficient expert scheduling across devices becomes the key factor, as balancing the workload between the CPU and GPU is essential for achieving decoding acceleration.

B. Insight 2: Skewed Experts Activation

Although modern MoE models often incorporate techniques such as auxiliary losses [11], [29], [30], bias [31], and expert routing [32] during training to encourage more balanced expert utilization, expert activation skew remains a persistent phenomenon. Our empirical analysis on the DeepSeek-V3 model across five domain-specific datasets demonstrates that this skew persists in practice: deeper layers exhibit more pronounced imbalance, while shallower layers show weaker skew, as shown in Figure 5. This observation provides experimental evidence to support dispatching experts based on their computational load.



(a) DS-10 Math (b) DS-50 Math (c) Qwen-10 Math (d) Qwen-50, Math

(e) DS-10 Text (f) DS-50 Text (g) Qwen-10 Text (h) Qwen-50 Text

Fig. 6: Token-wise Activation Heatmap. Heatmaps of expert activations during decoding across different MoE layers on various task types.'DS-10': DeepSeek-V3 layer 10.

C. Insight 3: Token-wise Activation Pattern

Through statistical analysis of expert activation data, we observe that during a complete decoding process, most layers exhibit consistent activation patterns, as illustrated in Figure 6. Figure 6 shows the expert activation patterns from token-wise perspective of two cutting-edge MoE models: DeepSeek-V3-671B [28] and Qwen3-235B [33]. Some patterns are noticeable: for instance, in the DeepSeek heatmaps, at layer 10, experts around ID 120 tend to be activated across different tasks, while at layer 50, experts around ID 152 and 208 are frequently activated. We randomly sampled 100 examples from each of five different task types for observation, and found that, except for shallow layers (e.g., before layer 10), where expert activations are relatively balanced, almost all subsequent layers exhibit certain consistent patterns. Similar phenomena also occur in the Qwen model. Due to space limitations, only four examples of each model are shown here. Therefore, we consider that expert activation at the token-wise level is to some extent predictable based on this observation.

IV. SMOE Design

A. Overview of SMOE

SMOE is designed around the core idea of achieving balanced CPU/GPU hybrid inference for ultra-large MoE models on a single consumer-grade GPU. SMOE executes the attention layers on the GPU, while the experts in the MoE layers are distributed across the CPU and GPU for parallel execution. It employs distinct optimization strategies for the prefilling and decoding phases according to their computational characteristics. For the prefilling phase, SMOE adopts an on-demand expert loading mechanism that leverages a hardware-aware expert allocation algorithm to identify which experts should be placed on the GPU for performance gains. Moreover, SMOE employs pipeline parallelism to further hide the expert loading latency. For the decoding phase, SMOE continues to exploit CPU/GPU collaborative computation to accelerate inference. To address the limited expert cache issue in deploying ultra-large models, SMOE proposes a novel token-wise prefetch

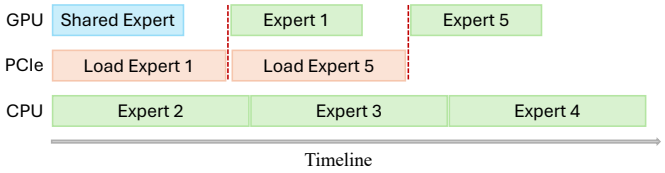


Fig. 7: Prefill Pipeline.

method that fully utilizes the computation time of each forward pass to hide transfer latency and maximize the number of experts prefetched. In addition, an efficient cache management scheme is designed to make efficient use of limited GPU memory and improve overall inference efficiency.

B. Hybrid CPU-GPU Execution for Prefilling

The prefilling phase is computation-intensive. In MoE layers, the number of tokens assigned to each expert varies — some experts process a large number of tokens (referred to as high-load experts), while others handle only a few (low-load experts). Given that GPUs typically offer several thousand to tens of thousands of times the computational throughput of CPUs, they are well-suited for executing high-load experts. Conversely, CPUs usually provide abundant and cost-effective memory resources, making them ideal for hosting low-load experts locally. Many existing MoE inference acceleration methods rely on expert activation statistics collected offline or runtime scheduling to manage expert placement. However, these pre-allocation methods often fail to fully distribute experts according to their load. A few high-load experts may be mistakenly assigned to the CPU, and due to the significant performance gap between CPU and GPU, this can lead to noticeable latency. This issue becomes increasingly severe as the prompt length grows.

To address this issue, SMOE adopts an on-demand expert loading strategy. During each MoE layer computation, experts are dynamically categorized into high-load and low-load groups based on the gating function's activation results. Using pre-profiled hardware capabilities and PCIe bandwidth, SMOE then determines an expert allocation policy that optimally distributes experts between CPU and GPU. This approach is both simple and effective. For the expert allocation policy, SMOE estimates the theoretical computation time of each expert on both GPU and CPU, as well as the data transfer time for the top-K high-load experts via PCIe. The optimal K is selected such that the total GPU computation time plus transfer time for these K experts is less than their CPU execution time. In addition, SMOE employs pipeline parallelism to further hide expert transfer latency between CPU and GPU, as illustrated in Figure 7. To efficiently manage memory during pipeline execution, we employ a ping-pong buffer mechanism to store experts, which significantly reduces GPU memory usage. Algorithm 1 illustrates the whole process.

Algorithm 1: Parallel CPU/GPU Expert Execution with Pipeline Optimization for Prefilling

Input: Expert Set: E;
GPU Time Function: $T_{GPU}(\cdot)$;
CPU Time Function: $T_{CPU}(\cdot)$;
PCIe transfer time per expert: T_{PCIe}

/* Expert allocation */ $C, G \leftarrow \varnothing$;
for e *in* E **do**
  if $T_{CPU}(e) > T_{GPU}(e) + T_{PCIe}$ **then**
    $G \leftarrow G \cup \{e\}$;
  else
    $C \leftarrow C \cup \{e\}$;

/* Parallel CPU–GPU execution */
parbegin
/* CPU thread */
for e *in* C **do**
  Execute e on CPU;
/* GPU thread with ping–pong buffering */
Initialize GPU buffers B_0, B_1;
$buf \leftarrow 0$;
AsyncLoad expert $G[1]$ from DRAM to B_{buf};
for $i = 1$ **to** $|G|$ **do**
  Launch GPU computation of expert $G[i]$;
  if $i < |G|$ **then**
    AsyncLoad expert $G[i+1]$ to B_{1-buf};
  Sync GPU computation of expert $G[i]$;
  $buf \leftarrow 1 - buf$;
parend
Sync CPU and GPU execution;

C. Hybrid CPU/GPU Execution for Decoding

The decoding stage is less compute-intensive than the prefilling. Still, it is markedly memory-bound: the KV-cache and other stateful buffers occupy a large fraction of GPU memory, constraining both the batch size and the cached expert capacity. Due to the sparse activation nature of MoE layers, applying the same on-demand expert loading strategy as in the prefilling phase leads to considerable data transfer latency, regardless of whether only a subset or all activated experts are transferred. This overhead becomes a dominant performance bottleneck, substantially slowing down inference. Alternatively, performing all expert computations on the CPU can avoid data transfer delays but causes the GPU to remain idle during CPU execution, resulting in poor resource utilization. Moreover, such an approach requires the CPU to have strong computational capability, which is expensive for consumer-grade devices.

To overcome these limitations, SMOE employs a hybrid CPU/GPU execution strategy to accelerate the decoding phase. The routed experts are distributed between CPU and GPU to

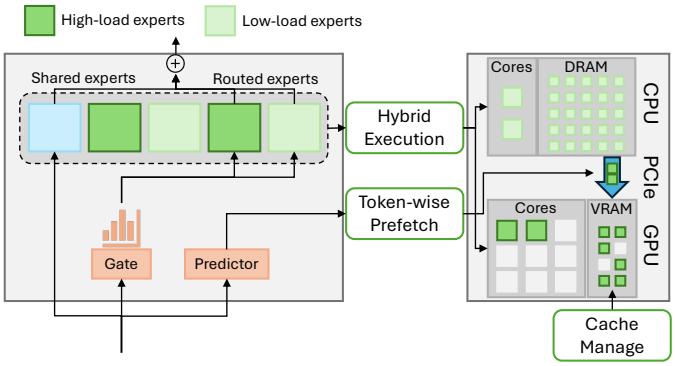


Fig. 8: Hybrid CPU-GPU Decoding.

execute concurrently, as illustrated in Figure 8. The shared experts (if exists) are always dispatched to the GPU. In theory, the best acceleration is achieved when the computational load of the CPU and GPU is balanced. However, reaching this balance presents a major challenge: how to schedule experts at runtime dynamically while hiding data transfer latency. To address this, we propose a novel token-wise prefetching mechanism along with an efficient expert cache management scheme, which together enable dynamic expert scheduling and effective overlap of data transfer with computation.

D. Token-wise Prefetch

For hybrid CPU/GPU inference, when GPU memory is sufficiently large or the model size is moderate, a considerable number of experts can be cached on the GPU. Since most existing MoE models still exhibit certain degrees of expert activation skew, it is feasible to identify frequently activated experts from historical statistics and keep them permanently resident on the GPU. In such cases, employing simple prefetching strategies for minor adjustments of expert placement can balance workloads and achieve near-optimal hybrid execution. However, as model size grows substantially, this assumption no longer holds. For instance, in DeepSeek-V3 (671B), which comprises 61 Transformer blocks (58 of which are MoE layers, each containing 256 experts and activating eight experts per token). On a 24 GB RTX 4090, approximately 10 GB of memory is consumed by the attention-related weights, while the KV cache for a 1K-length prompt requires around 1.5 GB. Caching eight experts per MoE layer with 4-bit quantization occupies roughly 10 GB, and using 8-bit weights would exceed 20 GB. Consequently, only a small fraction of experts (e.g., the top 3%) can reside on the GPU, covering merely a limited portion of total computation. This severe workload imbalance leads to suboptimal hardware utilization and degraded inference efficiency.

Therefore, to maintain high throughput on such large-scale models, it is crucial to design an effective prefetching mechanism that improves cache hit rates without introducing additional transfer overhead. This motivates the token-wise prefetching strategy proposed in SMOE, which dynamically predicts and loads the next-token experts during computation to fully overlap data transfer with ongoing execution. Traditional prefetching strategies, iteration-wise and layer-wise, are both insufficient to meet this requirement. The iteration-wise approach predicts all experts to be activated in the upcoming forward pass. It achieves relatively high accuracy in the shallow layers but leaves little time for data transfer. Conversely, in deeper layers, it provides more time but suffers from poor prediction accuracy. The layer-wise approach, on the other hand, predicts experts several layers ahead, achieving higher accuracy but with limited overlap time for data transfer.

To overcome these limitations, we propose a token-wise prefetching strategy. At each MoE layer, a lightweight predictor forecasts the experts likely to be activated by the next token and initiates prefetching immediately. Although its per-layer prediction accuracy is lower than that of layer-wise prefetching, it reserves an entire forward computation window, covering both non-MoE and other MoE layers, for data transfer. This design improves PCIe bandwidth utilization, enhances the tolerance for PCIe fluctuation and effectively hides transfer latency. Moreover, a high prediction accuracy is not required in the hybrid execution framework, since a portion of the experts is executed on the CPU. In addition, the token-wise approach offers greater flexibility. If certain layers exhibit low prediction accuracy, their prefetching can be selectively disabled, reallocating their available time budget to more accurate layers to improve overall cache hit rates. Other methods cannot achieve this level of adaptability: iteration-wise prefetching cannot redistribute the idle time of later layers to earlier ones, and layer-wise prefetching can only reassign time within the limited scope of its multi-layer prediction window. Figure 9 illustrates the token-wise prefetching strategy.

The predictor adopts an MLP with the same architecture as the MoE gate. It takes the current token's activation as input and produces outputs in the same format as the MoE gate, including the scores of all experts and the indices of the top-k selected experts. Owing to its lightweight design and minimal computational cost, the predictor's execution time is negligible compared to the overall inference process and has no significant impact on performance. The training of this predictor is also very simple, requiring only a small number of decoding samples (e.g., a few hundred). This process is performed offline, and due to the simplicity of the predictor's architecture, the training speed is fast. Naturally, the predictor can also continuously collect users' decoding data for online training to maintain better accuracy. In the subsequent experiments of this paper, the predictor is trained using 20 randomly selected samples from five datasets, and these training samples are excluded from all subsequent experiments.

E. Expert Cache Management

Due to the limited memory capacity of consumer-grade devices, an efficient cache management strategy is crucial, particularly when working with large-scale models. In the decoding stage, SMOE adopts a cache-prefetch strategy that determines the number of cached experts per layer based on the available GPU memory. In the prefilling, SMOE records

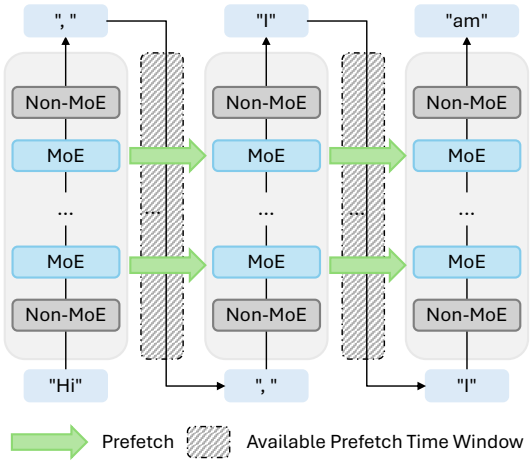


Fig. 9: Our Token-wise Prefetch.

the activation frequency of all experts. Once the prefilling ends, it allocates a predefined number of expert memory slots on the GPU and fills them with the most frequently activated experts as the initial cache. Subsequent cache updates are performed in-place by replacing existing experts without requiring the allocation of new memory. During decoding, the predictor begins its prediction immediately after obtaining the activation values from the current token's attention layer output. Once the computation of the MoE layer on the GPU is completed, the current prefetch task is submitted to an asynchronous transfer queue to update the cache of the corresponding layer. Meanwhile, the decoding process continuously collects expert activation statistics to support cache eviction. SMOE implements two eviction strategies:

- Fixed-N eviction, where the top-N predicted experts not already present in the cache replace the N least frequently activated experts, resulting in a fixed update count.
- Adaptive eviction, where all top-N predicted experts that are not in the cache are loaded into it, leading to a variable number of expert transfers (up to N).

Here, N is a tunable hyperparameter that can be adjusted according to device compute capability and PCIe bandwidth. Both strategies fall under the LFU (Least Frequently Used) policy, differing mainly in the amount of data transferred and their replacement preferences. Fixed-N eviction fixes the number of experts transferred each time, which means a fixed number of experts are updated in every step. This results in a more aggressive cache update strategy, while providing more stable PCIe utilization and memory usage. Such stability facilitates system monitoring and enables easier integration with future optimizations such as CUDA Graphs. In contrast, Adaptive eviction may transfer fewer experts, leading to a more conservative cache update behavior. In theory, it is better suited for models whose expert activation patterns exhibit significant skewness.

V. Implementation

This project is built upon KTransformers [12], which provides a flexible operator insertion framework that allows us to easily replace the original model operators. We keep the original KTrans computation structure, such as the attention modules, on the GPU, while modifying the MoE operator to enable selective execution of experts on the CPU. We also implemented a GPU operator for expert computation, along with a CUDA kernel for dequantizing experts cached in GPU memory. In addition, we implemented prefetching and cache management in C++ to further improve runtime efficiency.

VI. Evaluation

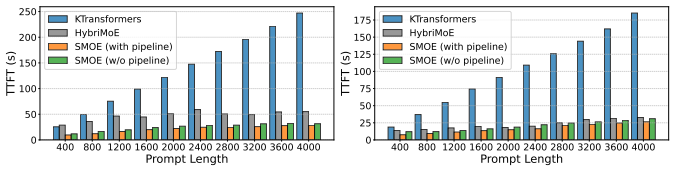
A. Experimental Setup

Hardware. Our experiments were conducted on a machine with the following hardware: AMD EPYC 7T83 (64 cores) $\times$ 2, 1 TB DRAM, NVIDIA RTX 4090 GPU with 24 GB HBM, and PCIe 4.0 $\times$16. In experiments, we used a single GPU and restricted the number of active CPU cores to ten or fewer to simulate the performance of a typical consumer-grade CPU.

Models. We evaluated SMOE with the 4-bit quantized DeepSeek-V3-0324 671B [34] and Qwen3-235B-A22B [33], two state-of-the-art MoE large language models from the open-source community. Compared with other state-of-the-art MoE offloading methods, SMOE is specifically optimized for scenarios involving ultra-large language models, where existing approaches often exhibit suboptimal performance.

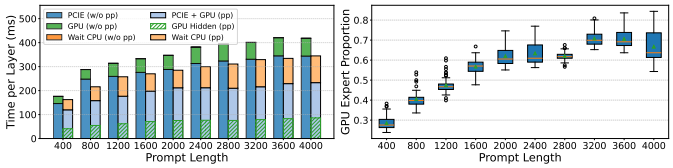
Datasets. For the prefilling phase, we used one dataset, LongBench [35], since the computational speed in this stage is independent of the dataset content. For the decoding phase, to comprehensively evaluate the effectiveness of SMOE, we adopted six datasets from different domains: the reading comprehension and question answering dataset TriviaQA [36], the machine translation dataset Flores-101 [37], the text summarization dataset XSum [38], the mathematical reasoning dataset GSM8K [39], the general question answering dataset Awesome ChatGPT Prompts [40], and the programming ability evaluation dataset HumanEval [41].

Baselines. We used KTransformers [12] and HybriMoE [13] as our baselines. KTransformers is an advanced framework for private deployment of MoE models on consumer-grade hardware. It executes the routed experts of MoE layers on the CPU while running the remaining modules on the GPU, achieving excellent inference performance on a single consumer-grade GPU. Built on the KTransformers framework, HybriMoE further optimizes the execution of routed experts in MoE layers. It adopts impact-driven prefetching (which is, in principle, a form of layer-wise prefetching) and score-aware caching strategies to enable hybrid execution optimization for experts, resulting in measurable inference speedups. However, the native implementation of HybriMoE only supports small- to medium-scale models such as Mixtral-8×7B (47B), Qwen2-57B, and DeepSeek-v2-lite-chat (16B). To facilitate a fair comparison, this paper reproduces the key techniques of HybriMoE



(a) DeepSeek-V3 (b) Qwen3

Fig. 10: Prefill End-to-End Performance. Time To First Token (TTFT) comparison on DeepSeek-V3 and Qwen3-235B.



(a) SMOE Prefill Time Breakdown (b) Proportion of Experts Processed on GPU in Prefilling.

Fig. 11: SMOE Prefill Time Breakdown and Proportion of Experts Processed on GPU in Prefilling.

and extends support to DeepSeek-V3 (671B) and Qwen3-235B. In our experiments, we do not use CUDA Graphs, so as to better highlight the performance benefits of the framework.

B. Prefilling End-to-End Performance

Figure 10 presents the Time To First Token (TTFT) comparison of SMOE with other methods on both DeepSeek-V3 and Qwen3 models under different prompt length for the prefilling stage. The baseline method, denoted as "KTrans", refers to the original KTransformers implementation and the "HybriMoE" denotes our reproduced and extended HybriMoE implementation that supports ultra-large models. The "Pipe" indicates our proposed pipeline optimization. Compared to the KTransformers baseline, SMOE demonstrate significant reductions in prefilling time, yielding up to 8.68× speedup on DeepSeek-V3 and 7× speedup on Qwen3. Compared to HybriMoE, SMOE also achieves up to 2.98× and 1.82× speedup on DeepSeek-V3 and Qwen3, respectively. KTransformers's TTFT increases significantly as the prompt length grows. This is mainly because KTransformers assigns all experts to the CPU for computation, whose computational capability is substantially weaker than that of the GPU. Similarly, HybriMoE's TTFT also increases with prompt length, but at a much slower rate than KTransformers since HybriMoE caches a subset of experts on the GPU, thereby alleviating the computational burden on the CPU. In contrast, SMOE achieves a shorter TTFT by employing more dynamic runtime expert allocation and pipelined execution strategies. Furthermore, the comparison of SMOE with and without pipeline operations reveals that SMOE 's pipeline design further reduces TTFT by overlapping data transfer with computation.

C. Prefill Time Breakdown

Figure 11a shows the time breakdown of SMOE in the prefill stage with and without pipeline optimization enabled. `PCIE` denotes the time spent transferring experts via PCIe, `GPU` refers to the time for GPU expert computation, and `Wait CPU` represents the time that the layer still needs to wait for the CPU to finish expert computation after excluding GPU and PCIe time (i.e., the time when the CPU becomes the bottleneck). From the experimental results, we can observe that SMOE achieves a certain degree of CPU/GPU load balancing during the prefill stage. The pipeline optimization effectively reduces the overall runtime by overlapping PCIe data transfer with GPU computation. Figure 11b shows the proportion of experts dispatched to the GPU during the prefilling for prompts of different lengths. As the figure illustrates, the number of experts processed by the GPU gradually increases with prompt length, since the number of workload-intensive experts grows. Our algorithm successfully identifies these experts and schedules them on the GPU, achieving accelerated prefilling.

D. Decoding End-to-End Performance

During the decoding, we compared our SMOE with KTransformers and HybriMoE on DeepSeek-V3 and Qwen3. SMOE employs a token-wise prefetching strategy, performing prefetching at every layer. SMOE-Flex disables prefetching for half of the layers with lower prediction accuracy, leaving time for the remaining layers. Throughout the experiments, a fixed-N replacement strategy was used for cache eviction to facilitate accurate measurement of the number of experts actually transferred. Additionally, during decoding, we consistently cache eight experts on the GPU, corresponding to approximately 3% of the total experts per layer. HybriMoE, as a representative hybrid CPU/GPU MoE inference method adopting a layer-wise prefetch strategy, achieves a certain degree of decoding speedup. However, it has not been evaluated on ultra-large models. We reproduced HybriMoE and extended it to support DeepSeek-V3 and Qwen3 as a baseline.

Table I presents the end-to-end decoding performance comparison of KTransformers, HybriMoE, and SMOE, on DeepSeek-V3 and Qwen3 models. In these experiments, KTransformers uses a CPU thread pool of size ten to handle expert computations. The experimental results for both HybriMoE and SMOE are obtained under their respective optimal prefetch expert counts. For HybriMoE, the optimal prefetch count is 1 for both models. For SMOE, it is 2 for DeepSeek-V3 and 3 for Qwen3, because the experts in Qwen3 are smaller than those in DeepSeek-V3. SMOE consistently outperforms KTransformers and HybriMoE on both DeepSeek-V3 and Qwen3 across all datasets. Under the optimal prefetch configurations, SMOE achieves up to 20.9% TPS improvement on DeepSeek-V3 and 16.2% on Qwen3 than KTransformers. Compared with HybriMoE, SMOE demonstrates more stable and higher throughput, indicating that its token-wise prefetching and scheduling strategy more effectively overlaps CPU, PCIe, and GPU execution. HybriMoE performs worse than KTransformers on some datasets, mainly because when

TABLE I: Decoding End-to-End Performance. Comparison of TPS on DeepSeek-V3 and Qwen3. The values in parentheses indicate the acceleration ratio compared to KTransformers.

Dataset	DeepSeek-V3				Qwen3			
	KTran	HybriMoE	SMOE	SMOE-Flex	KTran	HybriMoE	SMOE	SMOE-Flex
AC_Prompts	3.084	3.118	3.600 (+16.7%)	**3.727 (+20.9%)**	2.919	2.998	**3.279 (+12.4%)**	3.067 (+5.1%)
Flores-101	3.132	2.952	**3.399 (+8.5%)**	3.316 (+5.9%)	2.933	2.936	3.052 (+4.1%)	**3.156 (+7.6%)**
Gsm8k	3.224	3.013	3.462 (+7.4%)	**3.470 (+7.6%)**	2.960	2.963	**3.187 (+7.7%)**	3.110 (+5.1%)
Humaneval	3.248	2.949	3.468 (+6.8%)	**3.530 (+8.7%)**	3.019	2.988	**3.197 (+5.9%)**	3.059 (+1.3%)
TriviaQA	3.100	2.778	**3.364 (+8.5%)**	3.338 (+7.7%)	2.869	2.953	2.994 (+4.4%)	**3.219 (+12.2%)**
Xsum	3.188	2.862	**3.472 (+8.9%)**	3.238 (+1.6%)	2.798	2.908	**3.252 (+16.2%)**	3.067 (+9.6%)

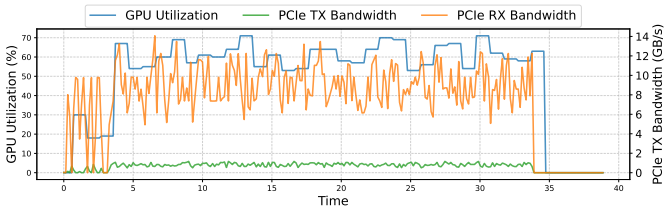


Fig. 12: GPU and PCIe Utilization of SMOE in the Decoding

applied to large models, the proportion of experts that can be cached on the GPU is very small (about 3% for DeepSeek-V3 and 6% for Qwen3). Moreover, HybriMoE's prefetching strategy does not effectively hide PCIe transfer latency, which leads to this performance degradation. In summary, compared with baseline methods, SMOE's token-wise prefetching approach is able to transfer more experts without introducing additional latency, achieving a better load balance between the CPU and GPU, and thus delivering better performance.

Figure 12 illustrates the GPU utilization and PCIe bandwidth usage of SMOE in a decoding process. SMOE achieves a GPU utilization of approximately 60%, primarily due to the GPU handling the attention computation, a subset of MoE experts, and the dequantization tasks preceding expert execution. Moreover, SMOE utilizes PCIe bandwidth up to 14 GB/s, which indicates an efficient use of the PCIe bus in a hybrid framework with frequent inter-device communication. Importantly, expert transfer incurs discrete communication overhead: adding one more expert results in an additional expert transfer at every layer, leading to a step-wise increase in data volume. As a result, the remaining PCIe bandwidth is insufficient to accommodate this increment. In practice, achievable PCIe bandwidth is further limited by hardware and system-level factors such as device contention and runtime variability. Nevertheless, given the fine-grained expert size ($<$10 MB) and the fact that SMOE transfers more experts than existing methods, the observed PCIe utilization can be considered relatively efficient, while PCIe bandwidth remains the main bottleneck.

E. Decode Time Breakdown

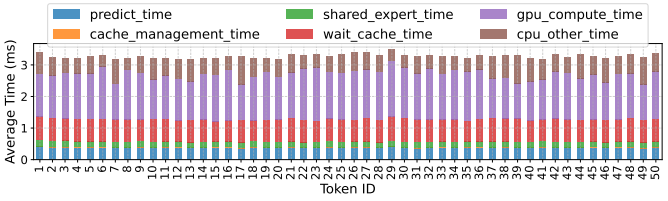
We conducted a detailed time breakdown experiment on the decoding stage of SMOE to validate the effectiveness of our approach. The prefetched expert number was two for both HybriMoE and SMOE. Figure 13 presents the time breakdown for expert computation tasks within the MoE layer under HybriMoE and SMOE frameworks, where:

- `predict_time`: Expert predictor runtime.
- `cache_management_time`: cache management time.
- `gpu_compute_time`: GPU routed experts runtime.
- `wait_cache_time`: GPU waiting for cache updates.
- `shared_expert_time`: GPU Shared experts runtime.
- `cpu_other_time`: The CPU time spent on routed expert computations that is *not* overlapped with GPU execution, i.e., the portion of CPU computation that becomes the critical-path bottleneck.

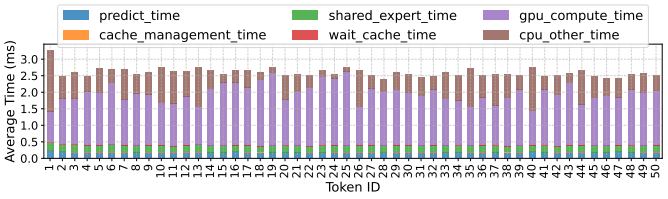
Experimental results show that, for the same amount of data transferred over PCIe, SMOE achieves a `wait_cache_time` that is nearly zero, whereas HybriMoE incurs approximately 1 ms of waiting time per token per MoE layer for expert cache updates. This indicates that the token-wise prefetching strategy of SMOE achieves near-perfect overlap between computation and communication. Moreover, the overhead of SMOE's cache management strategy is also negligible. Through dedicated microbenchmarks, we measured this overhead to be approximately 0.01 ms for DeepSeek-V3. This demonstrates that our cache management strategy is highly efficient.

F. Prefetch Study

Figure 14 compares the achieved throughput (TPS) and cache hit rate between the layer-wise prefetching method used in HybriMoE and the token-wise prefetching method employed by SMOE under different prefetch expert number configurations. It can be observed that, although the cache hit rate of the layer-wise method increases with the prefetch number, the acceleration gain it brings is consistently smaller than the incurred overhead, resulting in progressively poorer performance. In contrast, the token-wise prefetching employed by SMOE improves the cache hit rate in all six datasets when the prefetch expert number is one or two, accompanied by a corresponding increase in TPS. This demonstrates that our token-wise approach can more effectively utilize computation time for prefetching, particularly leveraging the computation time in the non-MoE layers of the model. For SMOE-Flex, the results indicate that reserving time for the layers with higher prefetch accuracy indeed increases the cache hit rate, as the

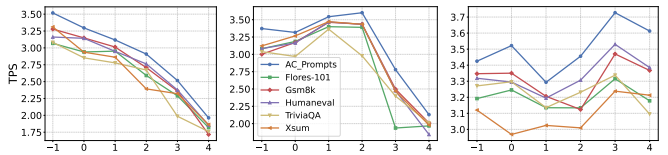


(a) HybriMoE

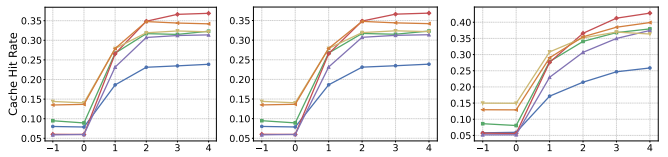


(b) SMOE

Fig. 13: MoE Expert Layer Time Breakdown of HybriMoE and SMOE by Tokens. CPU computing overlapped with GPU computing and shared expert computing.



(a) Layer-wise (Hybri+) (b) Token-wise (SMOE) (c) Token-wise (Flex)



(d) Layer-wise (Hybri+) (e) Token-wise (SMOE) (f) Token-wise (Flex)

Fig. 14: Comparison of TPS and Cache Hit Rate on Six Datasets for Layer-wise, SMOE, and SMOE-Flex Prefetching Methods. The x-axis is the number of prefetched experts: -1 denotes only the initial cache are used, 0 denotes prediction only (no prefetch). SMOE-Flex disables prefetching for half of the layers to leave time for the remaining layers.

remaining layers can prefetch up to three experts, compared to only one or two in the original SMOE.

Figure 15 shows the cache hit rate of HybriMoE and SMOE during decoding on the DeepSeek model. `Static` denotes the hit rate of the initial expert cache. For HybriMoE, the layer-wise prefetch strategy cannot effectively hide data transfer latency on ultra-large models, increasing the number of prefetched experts instead degrades performance. Therefore, we set its prefetch number to 1. The prefetch number is set to 2 for SMOE and 3 for SMOE-Flex. As shown in the figure, SMOE achieves a higher cache hit rate than HybriMoE because it updates one more expert per iteration. The hit rate of SMOE-Flex is computed only over the layers where

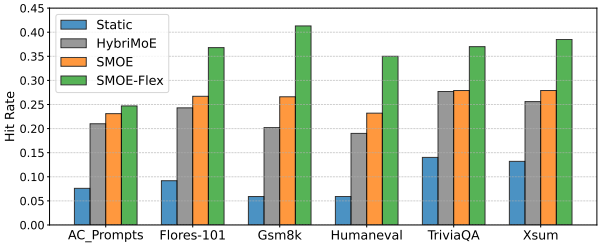


Fig. 15: Cache Hit Rate Comparison of HybriMoE, SMOE, and SMOE-Flex under Their Optimal TPS Performance

prefetching is enabled, and it is the highest since it updates the largest number of experts each time. Although SMOE-Flex attains a higher hit rate than SMOE, their overall performance is comparable. This is mainly because only about half of the layers in SMOE-Flex enable prefetching, which limits the achievable speedup. Overall, SMOE outperforms existing methods, while SMOE-Flex requires complex hyperparameter tuning to achieve favorable performance.

G. Hardware Sensitivity Analysis

Since the effectiveness of SMOE depends on hardware characteristics, we analyze its behavior under different CPU, GPU, and PCIe capabilities. For the CPU, when CPU expert computation time is comparable to that of the GPU, transferring experts via PCIe provides limited benefit and may not amortize the overhead. However, as batch size increases, the GPU's advantage becomes more pronounced, and SMOE's token-wise prefetching remains effective. When CPU performance is weaker, SMOE is particularly well-suited for acceleration.

Regarding PCIe, SMOE's token-wise prefetching offers one of the largest effective prefetch windows among hybrid inference methods, independent of bandwidth. While layer-wise methods may perform better when PCIe bandwidth is sufficiently high and transfers can be fully hidden within a single layer, PCIe performance is often variable in practice. We observe that layer-wise methods are more susceptible to transfer stalls, whereas token-wise prefetching better tolerates such variability. For the GPU, the primary bottleneck is HBM. Smaller HBM limits the expert cache, making the advantages of token-wise prefetching increasingly significant.

H. Model Accuracy

SMOE does not introduce any modifications or approximations to the model's computation graph. It merely distributes the computation of experts across the CPU and GPU. In theory, it should have no impact on model accuracy. This section experimentally verifies this claim. We randomly selected five samples from the GSM8K dataset

TABLE II: Perplexity comparison between SMOE and KTransformers.

Prompt	SMOE	KTrans
0	1.0731	1.0731
1	1.0603	1.0603
2	1.1256	1.1256
3	1.0856	1.0856
4	1.1366	1.1366

and monitored the decoding process of the DeepSeek model deployed under both the KTransformers and SMOE frameworks. During the experiments, the temperature was set to 1, and the token sampling strategies were disabled. Table II presents a comparison of perplexity, showing that the perplexity yield during decoding with SMOE is identical to that of KTransformers. These results demonstrate that SMOE is accuracy-preserving (i.e., lossless). With respect to quantization, studies [42], [43] have demonstrated that 4-bit quantization provides a favorable trade-off between model size and performance. Experiments by Zhao et al. [44] show that a 4-bit quantized DeepSeek-V3 incurs an average accuracy degradation of only 1.2% across multiple benchmarks. Accordingly, SMOE uses this model for evaluation.

VII. Scalability Analysis

Token-wise prefetching in SMOE scales well to multi-GPU and next-generation high-TPS systems. In multi-GPU environments, where inter-GPU communication is a bottleneck, it enables fine-grained, early expert transfers and overlaps communication with computation at the token level, reducing synchronization overhead and load imbalance. On next-generation platforms with larger HBM and high-bandwidth interconnects (e.g., NVLink and NVSwitch), it remains effective despite increased memory capacity, as ultra-large MoE models still impose memory pressure. It also benefits from extended comm–comp overlap and adapts well to heterogeneous experts and communication variability. Under dynamic routing, prediction accuracy may drop, but it still provides acceleration by partially balancing CPU/GPU workloads without noticeable overhead. Overall, token-wise prefetching complements hardware advances and is well suited for future large-scale, heterogeneous inference systems. SMOE is compatible with CUDA Graph via cudaLaunchHostFunc and multiple recorded graphs, but this adds complexity. To isolate SMOE's core benefits, we do not integrate CUDA Graph in this work.

VIII. Related Work

A. LLM Serving in Consumer-grade Devices

Recent efforts toward LLM serving on consumer devices have predominantly centered around the offloading paradigm, which can be broadly categorized into two types.

The first category focuses on high-throughput optimization. For instance, Klotski [45] hides data transfer latency by increasing the batch size and predicting hot–cold experts, though its effectiveness depends on sufficiently large batches. EMoE [46] adopts a similar strategy and likewise benefits from larger batch sizes. FlexGen [47] employs a zig-zag pipeline scheduling mechanism to optimize for high-throughput inference. MoE-Lightning [48], through Roofline model analysis, identifies throughput bottlenecks and proposes executing attention computation on the CPU while assigning MoE computation to the GPU. MegaScale-Infer [49] achieves high throughput by allocating prefilling and decoding across different devices. MoE-Gen [50] leverages module-based batching to improve parallelism, and MoE-Lens [51], similar to MoE-Lightning, separates attention and MoE computations for better utilization. These approaches are generally designed for evaluation-oriented scenarios, where maximizing throughput is more critical than minimizing latency.

The second category targets low-concurrency, low-latency applications. KTransformers [12], for example, places compute-intensive attention and head projection operations on the GPU, while offloading parameter-heavy but compute-light MoE layers to the CPU. Methods such as Pro-Gated MoE [15], MoE-Infinity [17], Fiddler [23], ProMoE [20], and SiDA-MoE [18] exploit the hot–cold expert phenomenon and activation similarity across MoE layers to predict expert usage, thereby enabling predictive expert scheduling for low-latency inference. Although PowerInfer [8] does not specifically target MoE models, it shares similar principles by leveraging offloading strategies to handle neuron-level computation adaptively. These approaches are better suited for latency-sensitive scenarios, such as intelligent customer service or personal AI assistants. The work presented in this paper also falls within this second category.

B. MoE Expert Activation Skewness

The uneven activation of experts in Mixture-of-Experts (MoE) architectures has persisted since MoE models were first adopted in large language models (LLMs). Numerous training studies have attempted to mitigate this issue through techniques such as auxiliary losses [11] and bias [31], yet none have achieved a fully satisfactory solution. For instance, ProMoE [20] analyzed several MoE-based models and observed that early MoE implementations, such as Switch-Transformer [29] and NLLB [52], exhibited severe load imbalance across experts. Later models, including DeepSeek [27] and Qwen [53], incorporated various balancing mechanisms that significantly improved the situation. However, according to our experimental observations, while these models achieve relatively balanced expert activation in shallow layers, imbalance still persists in deeper layers. MoE-Infinity [17] also reported that in Mixtral [26], the set of "hot experts" varies dynamically across different prompts during the decoding process. Similarly, Fiddler [23] statistically analyzed Mixtral and confirmed the presence of non-uniform expert utilization.

IX. Conclusion

In this work, we presented SMOE, a CPU/GPU hybrid MoE inference framework that enables efficient deployment of large and ultra-large models on a single consumer-grade GPU. By applying stage-aware optimizations to the prefill and decode phases, SMOE combines hybrid execution, pipelined data transfers, and a novel token-wise prefetching strategy with effective cache management to balance CPU and GPU workloads. Extensive evaluations demonstrate that SMOE consistently outperforms existing hybrid inference approaches. These results highlight the effectiveness and scalability of SMOE for future large-scale MoE inference under constrained GPU memory and heterogeneous system environments.

REFERENCES

[1] B. Zhang, B. Haddow, and A. Birch, "Prompting large language model for machine translation: A case study," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 41 092–41 110. [Online]. Available: https://proceedings.mlr.press/v202/zhang23m.html

[2] L. Wang, C. Lyu, T. Ji, Z. Zhang, D. Yu, S. Shi, and Z. Tu, "Document-level machine translation with large language models," *arXiv preprint arXiv:2304.02210*, 2023.

[3] F. Periti and S. Montanelli, "Lexical semantic change through large language models: a survey," *ACM Comput. Surv.*, vol. 56, no. 11, Jun. 2024. [Online]. Available: https://doi.org/10.1145/3672393

[4] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[5] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474*, 2022.

[6] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin *et al.*, "A survey on large language model based autonomous agents," *Frontiers of Computer Science*, vol. 18, no. 6, p. 186345, 2024.

[7] T. Guo, X. Chen, Y. Wang, R. Chang, S. Pei, N. V. Chawla, O. Wiest, and X. Zhang, "Large language model based multi-agents: A survey of progress and challenges," *arXiv preprint arXiv:2402.01680*, 2024.

[8] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, ser. SOSP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 590–606. [Online]. Available: https://doi.org/10.1145/3694715.3695964

[9] g gerganov, "gerganov/llama.cpp: Llm inference in c/c++," *https://github.com/ggml-org/llama.cpp*, 2023.

[10] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, no. 2, pp. 275–293, 2014.

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

[12] KVCache-AI, "Ktransformers: A flexible framework for experiencing cutting-edge llm inference optimizations," *https://github.com/kvcache-ai/ktransformers*, 2024.

[13] S. Zhong, Y. Sun, L. Liang, R. Wang, R. Huang, and M. Li, "Hybrimoe: Hybrid cpu-gpu scheduling and cache management for efficient moe inference," *arXiv preprint arXiv:2504.05897*, 2025.

[14] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, and M. Xu, "Edgemoe: Empowering sparse large language models on mobile devices," *IEEE Transactions on Mobile Computing*, 2025.

[15] R. Hwang, J. Wei, S. Cao, C. Hwang, X. Tang, T. Cao, and M. 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*, ser. ISCA '24. IEEE Press, 2025, p. 1018–1031. [Online]. Available: https://doi.org/10.1109/ISCA59077.2024.00078

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

[17] L. Xue, Y. Fu, Z. Lu, C. Sun, L. Mai, and M. K. Marina, "Moe-infinity: Efficient moe inference on personal machines with sparsity-aware expert cache," *arXiv preprint arXiv:2401.14361*, 2025.

[18] Z. Du, S. Li, Y. Wu, X. Jiang, J. Sun, Q. Zheng, Y. Wu, A. Li, H. Li, and Y. Chen, "Sida: Sparsity-inspired data-aware serving for efficient and scalable large mixture-of-experts models," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 224–238, 2024.

[19] X. He, S. Zhang, Y. Wang, H. Yin, Z. Zeng, S. Shi, Z. Tang, X. Chu, I. Tsang, and O. Y. Soon, "Expertflow: Optimized expert activation and token allocation for efficient mixture-of-experts inference," *arXiv preprint arXiv:2410.17954*, 2024.

[20] X. Song, Z. Zhong, and R. Chen, "ProMoE: Fast MoE-based LLM Serving using Proactive Caching," Oct. 2024, arXiv:2410.22134 [cs] Issue: arXiv:2410.22134. [Online]. Available: http://arxiv.org/abs/2410.22134

[21] S. Zhong, L. Liang, Y. Wang, R. Wang, R. Huang, and M. Li, "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.

[22] H. Huang, N. Ardalani, A. Sun, L. Ke, S. Bhosale, H.-H. Lee, C.-J. Wu, and B. Lee, "Toward efficient inference for mixture of experts," *Advances in Neural Information Processing Systems*, vol. 37, pp. 84 033–84 059, 2024.

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

[24] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.

[25] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling Laws for Neural Language Models," Jan. 2020, arXiv:2001.08361 [cs]. [Online]. Available: http://arxiv.org/abs/2001.08361

[26] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand, G. Lengyel, G. Bour, G. Lample, L. R. Lavaud, L. Saulnier, M.-A. Lachaux, P. Stock, S. Subramanian, S. Yang, S. Antoniak, T. L. Scao, T. Gervet, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed, "Mixtral of Experts," Jan. 2024, arXiv:2401.04088 [cs]. [Online]. Available: http://arxiv.org/abs/2401.04088

[27] A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[28] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[29] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: scaling to trillion parameter models with simple and efficient sparsity," *J. Mach. Learn. Res.*, vol. 23, no. 1, pp. 120:5232–120:5270, 2022.

[30] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[31] L. Wang, H. Gao, C. Zhao, X. Sun, and D. Dai, "Auxiliary-loss-free load balancing strategy for mixture-of-experts," *arXiv preprint arXiv:2408.15664*, 2024.

[32] Y. Zhou, T. Lei, H. Liu, N. Du, Y. Huang, V. Zhao, A. M. Dai, Q. V. Le, J. Laudon *et al.*, "Mixture-of-experts with expert choice routing," *Advances in Neural Information Processing Systems*, vol. 35, pp. 7103–7114, 2022.

[33] Q. Team, "Qwen3 technical report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388

[34] DeepSeek-AI, "Deepseek-v3 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2412.19437

[35] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, Y. Dong, J. Tang, and J. Li, "Longbench: A bilingual, multitask benchmark for long context understanding," 2023.

[36] M. Joshi, E. Choi, D. S. Weld, and L. Zettlemoyer, "Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension," *arXiv preprint arXiv:1705.03551*, 2017.

[37] N. Goyal, C. Gao, V. Chaudhary, P.-J. Chen, G. Wenzek, D. Ju, S. Krishnan, M. Ranzato, F. Guzmán, and A. Fan, "The flores-101 evaluation benchmark for low-resource and multilingual machine translation," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 522–538, 2022.

[38] S. Narayan, S. B. Cohen, and M. Lapata, "Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization," *ArXiv*, vol. abs/1808.08745, 2018.

[39] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[40] F. K. Akın, "Huggingface datasets: fka/awesome-chatgpt-prompts," *https://huggingface.co/datasets/fka/awesome-chatgpt-prompts*, 2023.

[41] M. Chen, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[42] T. Dettmers and L. Zettlemoyer, "The case for 4-bit precision: k-bit inference scaling laws," in *International Conference on Machine Learning*. PMLR, 2023, pp. 7750–7774.
[43] Z. Liu, C. Zhao, H. Huang, S. Chen, J. Zhang, J. Zhao, S. Roy, L. Jin, Y. Xiong, Y. Shi *et al.*, "Paretoq: Scaling laws in extremely low-bit llm quantization," *arXiv preprint arXiv:2502.02631*, 2025.
[44] E. Zhao, Y. Shen, S. Shi, J. Huang, Z. Chen, N. Wang, S. Xiao, J. Zhang, K. Wang, and S. Lian, "Quantitative analysis of performance drop in deepseek model quantization," *arXiv preprint arXiv:2505.02390*, 2025.
[45] Z. Fang, Y. Huang, Z. Hong, Y. Lyu, W. Chen, Y. Yu, F. Yu, and Z. Zheng, "Klotski: Efficient Mixture-of-Expert Inference via Expert-Aware Multi-Batch Pipeline," Feb. 2025, arXiv:2502.06888 [cs]. [Online]. Available: http://arxiv.org/abs/2502.06888
[46] S. Tairin, S. Mahmud, H. Shen, and A. Iyer, "eMoE: Task-aware Memory Efficient Mixture-of-Experts-Based (MoE) Model Inference," Mar. 2025, arXiv:2503.06823 [cs]. [Online]. Available: http://arxiv.org/abs/2503.06823
[47] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, Z. Xie, B. Chen, C. Barrett, J. E. Gonzalez, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," Jun. 2023, arXiv:2303.06865 [cs]. [Online]. Available: http://arxiv.org/abs/2303.06865
[48] S. Cao, S. Liu, T. Griggs, P. Schafhalter, X. Liu, Y. Sheng, J. E. Gonzalez, M. Zaharia, and I. 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*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 715–730. [Online]. Available: https://dl.acm.org/doi/10.1145/3669940.3707267
[49] R. Zhu, Z. Jiang, C. Jin, P. Wu, C. A. Stuardo, D. Wang, X. Zhang, H. Zhou, H. Wei, Y. Cheng, J. Xiao, X. Zhang, L. Liu, H. Lin, L.-W. Chang, J. Ye, X. Yu, X. Liu, X. Jin, and X. Liu, "MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism," Apr. 2025, arXiv:2504.02263 [cs]. [Online]. Available: http://arxiv.org/abs/2504.02263
[50] T. Xu, L. Xue, Z. Lu, A. Jackson, and L. Mai, "MoE-Gen: High-Throughput MoE Inference on a Single GPU with Module-Based Batching," Mar. 2025, arXiv:2503.09716 [cs]. [Online]. Available: http://arxiv.org/abs/2503.09716
[51] Y. Yuan, L. Ma, and N. Talati, "MoE-Lens: Towards the Hardware Limit of High-Throughput MoE LLM Serving Under Resource Constraints," Apr. 2025, arXiv:2504.09345 [cs]. [Online]. Available: http://arxiv.org/abs/2504.09345
[52] M. R. Costa-Jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard *et al.*, "No language left behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.
[53] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

ACKNOWLEDGMENT

This work is supported by National Key R&D Program of China under Grant No. 2024YFA1012700. It is also funded by the NSFC Project (No. 62306256) and the Natural Science Foundation of Guangdong Province (No. 2025A1515010261).

Artifact Description (AD)

X. Overview of Contributions and Artifacts

A. Paper's Main Contributions

C_1 SMOE achieves prefill acceleration by offloading high-load experts layer by layer and optimizing the pipeline.
C_2 It enables decode acceleration by performing token-wise prefetching to promptly update the GPU expert cache.

B. Computational Artifacts

A_1 https://doi.org/10.6084/m9.figshare.31353823
A_2 https://github.com/rzhang772/SMOE

Artifact ID	Contributions Supported	Related Paper Elements
A_1, A_2	C_1	Figure 10-11
A_1, A_2	C_2	Tables I

XI. Artifact Identification

A. Computational Artifact A_1

Relation To Contributions

A_1 is the DOI for all code resources of SMOE, and A_2 is its GitHub repository. SMOE employs specialized optimizations for prefilling and decoding, thereby achieving inference acceleration (i.e., C_1 and C_2).

Expected Results

The code in A_2 should achieve faster inference speeds on the Deepseek-V3-671B 4-bit quantized model and the Qwen3-235B 4-bit quantized model compared to KTransformers (https://github.com/kvcache-ai/ktransformers) and HybriMoE (https://github.com/PKU-SEC-Lab/HybriMoE).

Expected Reproduction Time (in Minutes)

The experiments were conducted within a Docker container. Installing the Docker image takes approximately 15 minutes. After the image is installed, the C++ source code needs to be compiled, with the initial compilation taking about 20 minutes. Testing different prompt samples takes between 3 to 10 minutes, depending on the input length; model loading accounts for a significant portion of this time, and the exact runtime varies based on the number of tokens.

Artifact Setup (incl. Inputs)

Hardware: AMD EPYC 7T83 (64 cores), 1 TB DRAM, NVIDIA RTX 4090 with 24 GB HBM, and PCIe 4.0 $\times$16.

Software: The system requires Ubuntu 24.04 or a newer version, along with Docker, a Python runtime environment, and a C++ compilation and execution environment. The required Python version and package dependencies are specified in the `requirements.txt` file in the GitHub repository.

Datasets / Inputs: Different datasets were used in the experiments to evaluate SMOE's acceleration performance across different stages. See the "Datasets" subsection in Section VI.A for details.

Installation and Deployment: We recommend installing Ubuntu (or another Linux distribution) and Docker. The specific Docker image requirements are provided in the Dockerfile. Inside the container, first verify that both the Python and C++ environments are properly set up. Then, run the install.sh script to install Python dependencies and compile the project—this step requires CMake. Once the installation completes successfully, you can proceed to run the experiments.

Artifact Execution

Once the required datasets for the experiments are prepared, you can start testing immediately. The code already includes logic for measuring inference speed. After each inference run, it will output the prefill time and throughput, as well as the decode time and throughput.

Artifact Analysis (incl. Outputs)

Each experiment run should first output the generated decode content, followed by a summary that includes:

- Number of prefill tokens, prefill time, and average prefill speed (tokens/second).
- Number of decode tokens, decode time, and average decode speed (tokens/second).