MoE-Balance: Unleashing the Full Potential of CPU-GPU Heterogeneity for Efficient MoE Inference

Tiantian Lin[1,2], Xiaohang Wang[1,2,*], Yingtao Jiang[3], Amit Kumar Singh[4] and Kui Ren[1,2]
[1]State Key Laboratory of Blockchain and Data Security, Zhejiang University, Hangzhou, China
[2]Hangzhou High-Tech Zone (Binjiang), Institute of Blockchain and Data Security, Hangzhou, China
[3]University of Nevada, Las Vegas, USA [4]University of Essex, Colchester, UK

Abstract

Sparse Mixture-of-Experts (MoE) models offer a powerful way to increase the capacity of large language models without proportionally increasing computation. Yet, their extremely large expert weight parameters create severe memory challenges for consumer-grade GPUs and edge devices. Existing offloading and hybrid CPU–GPU approaches fall short in fully utilizing CPU resources, and their coarse-grained scheduling and poorly overlapped CPU-GPU data transfers leave CPU cores frequently idle, thus causing pipeline bubbles and exacerbating overall system inefficiency. We present MoE-Balance, a system that efficiently utilizes heterogeneous CPU-GPU resources for MoE inference. The key innovations include a fine-grained dynamic scheduler that distributes expert tasks based on real-time load, and a cross-layer prefetching and chunked transfer mechanism that overlaps computation with weight movement via PCIe. Experiments on Mixtral-8×7B show that MoE-Balance achieves 1.6× higher throughput and 37.5% lower end-to-end latency compared with state-of-the-art hybrid baselines, demonstrating strong performance gains under memory-constrained settings.

Keywords

MoE inference, Weight offloading, CPU-GPU hybrid execution

ACM Reference Format:
Tiantian Lin[1,2], Xiaohang Wang[1,2,*], Yingtao Jiang[3], Amit Kumar Singh[4] and Kui Ren[1,2]. 2026. MoE-Balance: Unleashing the Full Potential of CPU-GPU Heterogeneity for Efficient MoE Inference. In *63rd ACM/IEEE Design Automation Conference (DAC '26), July 26–29, 2026, Long Beach, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3770743.3803978

1 Introduction

Sparse Mixture-of-Experts (MoE) layers replace the Transformer Feed-Forward Network (FFN) with many experts and use a gating network to activate only a few per token, enabling large parameter growth without proportional compute increase [14, 15], as illustrated in Figure 1. However, the massive total parameter count of MoE models, especially the expert weights, poses severe challenges for deployment on memory-constrained platforms such as *consumer-grade* GPUs (e.g., GTX 3090 or 4090 series, with 16~24 GB VRAM). For example, the Mixtral 8×22B model requires over 280 GB of memory for the expert FFN parameters [3], which is 4-5× higher than the memory requirements of dense models with comparable FLOPs [2].

* Xiaohang Wang is the corresponding author.

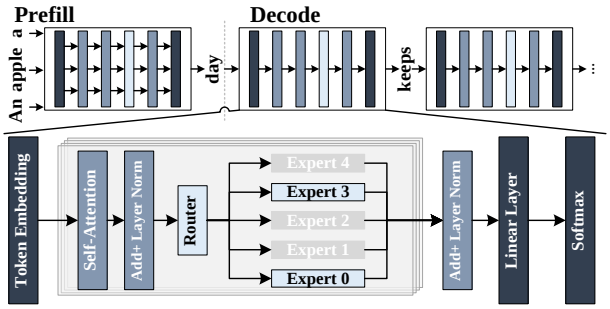


Figure 1: An example of the decoder-only MoE-based LLM inference procedure with top-2 experts.

Traditional solutions such as weight offloading [1] store expert weights in CPU memory and load them to the GPU on demand, but this becomes highly inefficient. As shown in Table 1, GPU computation per decoding step is short, whereas PCIe transfers take several times longer, making I/O latency the main bottleneck and causing frequent GPU idling.

Weight prefetching methods (e.g., MoE-Infinity [17]) attempt to hide this latency, but they largely overlook CPU compute capacity. Subsequent hybrid CPU–GPU methods (e.g., HybriMoE [21], Fiddler [13]) improve utilization but still suffer from two key limitations: i) coarse-grained or static expert scheduling that cannot adapt to dynamic expert activation, causing CPU–GPU execution time imbalance; ii) high PCIe transfer latency that dominates execution, preventing full overlap of CPU and GPU computation.

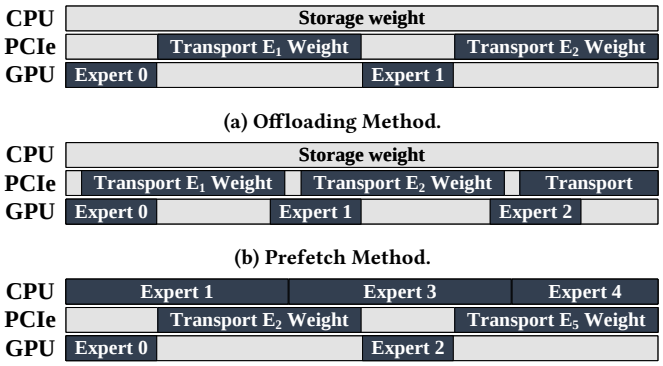
Table 1: Breakdown of execution time.

	Mixtral-8×7B [12]	DeepSeek-V2-Lite-Chat [4]
CPU Block Execution	11.1	2
GPU Block Execution	1	1
Expert Transfer	124	73
Note: GPU block execution of Mixtral-8x7B is five times of DeepSeek-V2-Lite-Chat.		

To overcome these challenges, we propose the **MoE-Balance System**, which fully leverages heterogeneous CPU-GPU resources for MoE inference. The core innovation lies in a dynamic, adaptive scheduling and weights transfer framework, tightly coupling scheduling with data weights to achieve optimal load balancing between CPU and GPU. In summary, the main contributions of this work are summarized as follows:

- **Performance Model for Heterogeneous MoE Inference**: We build a quantitative execution-time model that captures stable relationships among CPU compute, GPU compute, and expert weight transfer, providing the foundation for dynamic CPU–GPU scheduling.
- **Runtime-Aware Dynamic Load Balancing**: We design a fine-grained scheduler that allocates expert computation across CPU and GPU based on real-time expert load, aiming to equalize per-token execution time and maximize heterogeneous utilization with minimal overhead.

This work is licensed under a Creative Commons Attribution 4.0 International License.
DAC '26, Long Beach, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3803978



(a) Offloading Method.

(b) Prefetch Method.

(c) CPU-GPU Co-computation Method.

Figure 2: Traditional MoE Deployment Methods.

- **Cross-Layer Prefetching with Chunked Transfers**: To prevent scheduler stalls from large weight movements, we introduce cross-layer prefetching and dynamic chunked transfers, enabling weight loading to overlap with computation and ensuring smooth runtime scheduling.

We evaluate MoE-Balance on representative MoE models, including Mixtral-8×7B [12] under varying GPU memory budgets (16 GB and 24 GB) and CPU logical threads. Compared with state-of-the-art baselines, MoE-Balance achieves 33.5×, 2.3×, 3.7×, and 1.6× higher throughput over MoE-Infinity, AdapMoE, Fiddler, and HybriMoE, respectively, and reduces end-to-end latency by 92%, 68%, 77%, and 37.5% over these aforementioned methods. These results, consistent across all tested hardware configurations, demonstrate that MoE-Balance effectively balances CPU–GPU workloads and delivers robust performance gains in memory-constrained settings.

2 Background and Motivation

2.1 Memory Challenges

MoE models excel in distributed GPU clusters, but running them on a single consumer-grade GPU faces major hurdles. The primary bottleneck is memory. Due to their large parameter counts, it can quickly exceed the VRAM of typical consumer-grade GPUs (16~48 GB), even after FP16 quantization, as shown in Table 2. This gap makes single-GPU deployment impractical without compression techniques such as quantization, which may reduce model accuracy.

Table 2: FP16 Storage Requirements of MoE Models.

Model	Total Parameters	Estimated FP16 Weight Storage
Mixtral-8×7B	46.7 B	~ 94 GB
Mixtral-8×22B	141 B	~ 282 GB
DBRX	132 B	~ 264 GB
DeepSeek-V2	236 B	~ 472 GB

2.2 Traditional MoE Deployment Methods

To deploy large-scale MoE models on consumer-grade GPUs, traditional methods primarily target GPU memory limitations, followed by system-level strategies to optimize performance and latency. The main capacity-oriented approaches are offloading and quantization.

Offloading. Model weights are stored in CPU DRAM [10, 16, 22], with the GPU fetching needed weights on demand, such as Mixtral-Offloading [6]. This reduces GPU memory pressure but introduces significant PCIe transfer overhead (Figure 2a).

Quantization. Reducing weight precision from FP16 to INT8 or INT4 [5, 7, 8] lowers memory and transfer costs and improves throughput, at the cost of potential accuracy loss.

Advanced strategies address I/O bottlenecks via prefetching and CPU–GPU co-computation as follows.

Weight Prefetching. By predicting future expert usage, weights are asynchronously moved from CPU to GPU to improve cache hits and reduce on-demand communication (Figure 2b) [10, 18]. MoE-Infinity [17] is a Sparsity-Aware Expert Cache to improve cache hits, while AdapMoE [20] adapts active expert counts to reduce on-demand loading and communication overhead.

CPU–GPU Co-computation. CPU computation reduces GPU load and PCIe contention (Figure 2c). Methods like Fiddler [13] statically assign experts, whereas HybriMoE [21] dynamically schedules experts.

2.3 Motivation

This section presents experimental observations from MoE inference conducted on consumer-grade GPU-CPU heterogeneous devices. We analyze the limitations of existing approaches and reveal critical performance bottlenecks and optimization opportunities.

CPU–GPU Execution Time Imbalance. Measurements on heterogeneous CPU–GPU reveal that current MoE inference frameworks underuse CPU compute resources. Figure 3 breaks down the execution times of CPU and GPU, as well as the overall per-layer MoE processing time, of several previous methods. For Mixtral-Offloading [6], the CPU spends most of its time idle, waiting for weight transfers from memory. Prefetching strategies in MoE-Infinity [17] and AdapMoE [20] reduce this idle time partially, but the CPU still cannot fully participate in computation. Fiddler [13] statically assigns experts to devices, which cannot adapt to dynamic expert activation, resulting in severe CPU–GPU load imbalance. HybriMoE [21] dynamically schedules experts considering transfer times, but the high PCIe transfer latency dominates, leaving the CPU underutilized and preventing full load balance. It does not consider weight prefetching and chunked transfer to overlap computation and communication and thus lead to pipeline bubbles caused by high PCIe transfer latency.

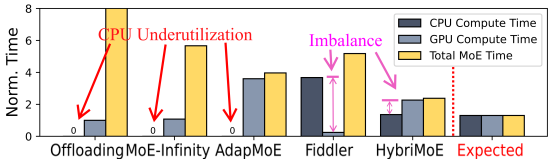


Figure 3: Comparison of MOE Total Time and CPU/GPU Computation Time of Different Methods.

These observations motivate our approach, MoE-Balance, which explicitly targets balanced CPU–GPU workloads while minimizing PCIe-transfer-induced stalls, maximizing heterogeneous resource utilization. MoE-Balance tries to balance CPU and GPU execution times to improve overall inference performance. The framework first constructs a performance model for all experts, which informs a dynamic allocation strategy that redistributes expert tasks across devices. To further reduce idle time and hide PCIe communication latency, MoE-Balance incorporates cross-layer prefetching and chunked weight transfers, ensuring CPUs are actively utilized and GPU stalls are minimized.

3 MoE-Balance Design

3.1 Overview

MoE-Balance introduces an intelligent scheduling framework to enable efficient CPU-GPU collaboration (Figure 4).

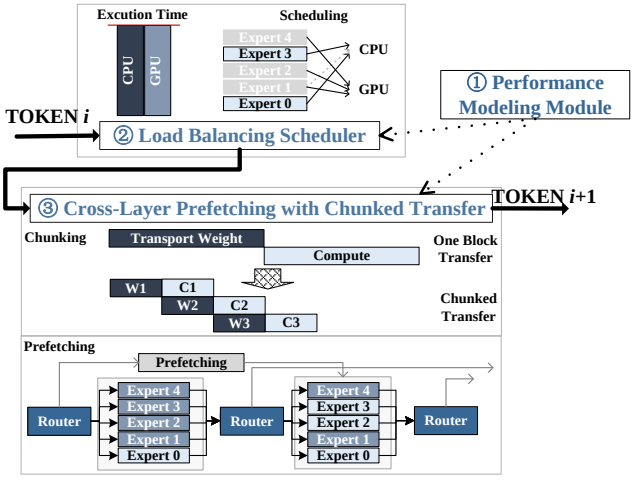


Figure 4: Overview of MoE-Balance.

Performance Modeling Module. At system initialization, all experts are profiled to measure CPU/GPU block execution times and weight transfer latency. This builds an accurate performance model that informs subsequent scheduling, ensuring expert execution times are predictable and helping to reduce bubbles. These metrics form the foundation for subsequent scheduling decisions and need to be measured only once under the same hardware and model configuration.

Execution-Time Equalization Scheduler. Using the performance model and real-time workload, it dynamically assigns experts to devices. An initial allocation prioritizes hot experts on GPUs while keeping CPUs active, and a runtime adaptive adjustment migrates experts to correct execution time discrepancies. This transforms CPUs from passive storage into active computing resources, improving throughput.

Cross-Layer Prefetching with Chunked Transfer. Leveraging inter-layer dependencies, it predicts future expert usage via the gate network and prefetches weights proactively. Large experts are transferred in chunks asynchronously, overlapping computation and communication to maximize PCIe bandwidth and minimize GPU idle time.

This framework enables MoE-Balance to fully exploit heterogeneous resources and achieve high-efficiency MoE inference.

3.2 Performance Modeling Module

To enable efficient expert scheduling across heterogeneous devices, this module constructs a unified latency prediction model that estimates computation time of each expert on CPU and GPU, as well as the weight transfer time between devices. The performance model is built once during system initialization, combining empirical measurements with regression-based modeling to provide quantitative foundations for subsequent load balancing and scheduling decisions.

In practice, MoE inference often exhibits highly consistent execution behaviors across experts. We measured the execution times of MoE components during inference for two representative models, Mixtral-8×7B [12] and DeepSeek-V2-Lite-Chat [4], with model configurations summarized in Table 4. Table 3 reports the coefficient of variation (CV [11]) for CPU execution time, GPU execution time, and Weight Transfer time across experts, where CV measures the

Table 3: Runtime Stability Across Layers.

Mixtral-8×7B			
Metrics	**CPU Execution Time**	**GPU Execution Time**	**Weight Transfer Time**
CV	0.01	0.06	0.003
DeepSeek-V2-Lite-Chat			
Metrics	**CPU Execution Time**	**GPU Execution Time**	**Weight Transfer Time**
CV	0.015	0.05	0.002

relative dispersion of each metric. The consistently low CV values indicate that these runtime behaviors are stable across experts, which supports building a unified latency prediction model.

Building on this observation, each expert is sequentially loaded and executed multiple times on both CPU and GPU to collect latency samples. To reduce stochastic variations, each measurement is repeated N times, and the average execution times are calculated as:

$$T_{\text{CPU}}(E_i) = \frac{1}{N}\sum_{k=1}^{N} T'_{\text{CPU}}(E_i,k), T_{\text{GPU}}(E_i) \quad = \frac{1}{N}\sum_{k=1}^{N} T'_{\text{GPU}}(E_i,k) \tag{1}$$

where $T'_{\text{CPU}}(E_i,k)$ and $T'_{\text{GPU}}(E_i,k)$ denote the measured execution time of expert E_i on CPU and GPU during the k-th trial (token k), respectively.

The module also measures the weight transfer time $T_{trans}(E_i)$ for each expert between CPU and GPU. All averaged computation and transfer times are stored in a performance mapping table `PerfTable`. During inference, the scheduler queries this table to obtain predicted latencies for informed expert placement under real-time workload conditions.

Through this procedure, MoE-Balance builds an accurate, low-overhead performance model that forms the foundation for load balancing and cross-layer prefetching, ensuring reliable and performance-aware expert scheduling across heterogeneous hardware.

3.3 Execution-Time Equalization Scheduler

The goal of the scheduler is to balance expert execution time across heterogeneous devices under GPU memory constraints. Let C_l and $\mathcal{G}_l$ denote the experts placed on the CPU and GPU at layer l, respectively. Given token routing results, where each expert E_i processes n_i tokens, the corresponding execution times on the two devices are:

$$T_l^{\text{CPU}} = \sum_{E_i \in C_l} T'_{\text{CPU}}(E_i, n_i), \quad T_l^{\text{GPU}} = \sum_{E_i \in \mathcal{G}_l} T'_{\text{GPU}}(E_i, n_i) \tag{2}$$

The scheduler aims to minimize the execution-time imbalance:

$$\min_{C_l, \mathcal{G}_l} \left| T_l^{\text{CPU}} - T_l^{\text{GPU}} \right| \tag{3}$$

subject to the GPU memory constraint:

$$|\mathcal{G}l| \le G\text{max}, \quad G_{\text{max}} = \frac{(1-\rho)M_{\text{GPU}}}{S_{\text{expert}}} \tag{4}$$

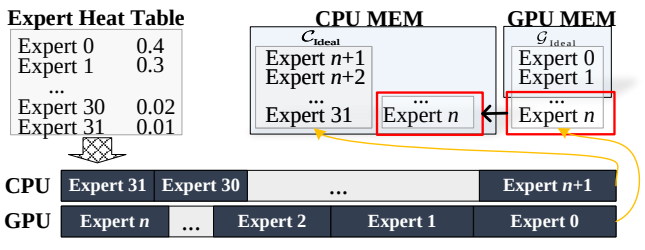
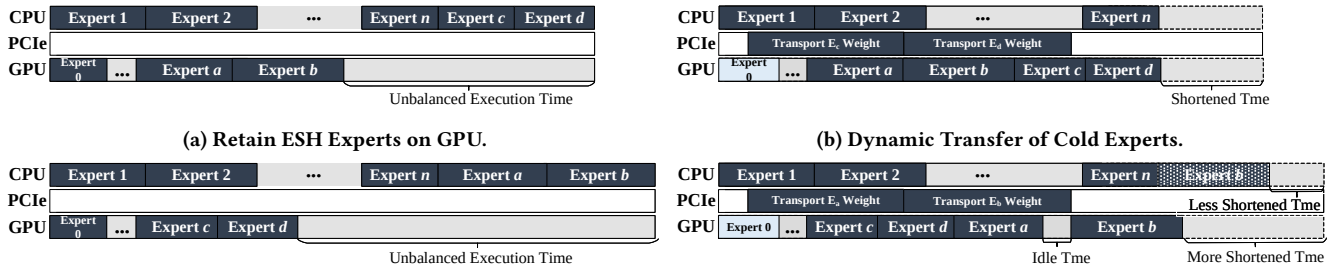


Figure 5: Ideal allocation



(a) Retain ESH Experts on GPU.

(b) Dynamic Transfer of Cold Experts.

(c) Retain Cold Experts on GPU.

(d) GPU Idle Due to Insufficient Load.

Figure 6: Illustration of ESH Expert Priority and Dynamic GPU-CPU Scheduling.

where $G_{\max}$ is the number of experts that can reside on the GPU, M_{GPU} is total GPU memory, ρ is the reserved memory ratio and S_{expert} the size of a single expert. To solve this problem, the following algorithm is proposed.

A. Initial Allocation Strategy

Step 0: Define Extremely Super Hot (ESH) Experts. Before allocation, experts are categorized based on usage frequency. An expert E_i is considered *Extremely Super Hot (ESH)* if its workload proportion exceeds a threshold β:

$$E_i \in \text{ESH} \quad \text{if} \quad \frac{w_i}{\sum_j w_j} > \beta \tag{5}$$

where w_i is the total number of tokens routed to E_i.

Step 1: Construct Ideal CPU-GPU Allocation. Let $\mathcal{C}_{\text{Ideal}}$ and $\mathcal{G}_{\text{Ideal}}$ denote the ideal expert sets for CPU and GPU, balancing expected execution time:

$$\widetilde{T}_0^{\text{CPU}} = \sum_{E_i \in \mathcal{C}_{\text{Ideal}}} T_{\text{CPU}}(E_i) \approx \sum_{E_i \in \mathcal{G}_{\text{Ideal}}} T_{\text{GPU}}(E_i) = \widetilde{T}_0^{\text{GPU}} \tag{6}$$

where $\widetilde{T}_0^{\text{CPU}}$ and $\widetilde{T}_0^{\text{GPU}}$ are the total estimated execution time of CPU and GPU at layer *0*, respectively. GPU allocations prioritize hot experts to reduce transfer overhead, as shown in Figure 5. In the figure, we assume experts are sorted by decreasing hotness, and it depicts the $\mathcal{C}_{\text{Ideal}}$ and $\mathcal{G}_{\text{Ideal}}$ allocation that balances expected execution times without considering memory constraints.

Step 2: Partition Experts Under Memory Constraints. Due to limited GPU memory ($G_{\max}$), some experts in the $\mathcal{G}_{\text{Ideal}}$ need to assigned to the CPU, as shown in Figure 5. Thus, we divide all experts $\mathcal{E}_{\text{all}}$ into GPU-resident $\mathcal{G}_{\text{stay}}$ and CPU-resident $\mathcal{C}_{\text{stay}}$, with $|\mathcal{G}_{\text{stay}}| < G_{\max}$. All ESH experts are placed on the GPU first. Remaining GPU capacity is filled with other experts in ascending order of hotness. This priority design is motivated by the observation that ESH experts typically process the majority of input tokens and dominate GPU execution time. Keeping them resident on the GPU maximizes overlap between computation and data transfer.

As shown in Figure 6, suppose experts $E_a, E_b, E_c, E_d \in \mathcal{G}_{\text{Ideal}}$, but two of them need to be placed on the CPU temporarily due to GPU memory constraints. E_a and E_b are ESH, so they remain on the GPU (Figure 6a). The scheduler dynamically transfers E_c and E_d to the GPU (Figure 6b), overlapping their transfer with GPU-resident computation and evicting cold experts (e.g., E_0) if necessary.

If instead E_c and E_d are retained on the GPU (Figure 6c), E_a and E_b should be transferred, but the lighter GPU load cannot fully hide the transfer time (Figure 6d), causing idle time. Even computing E_b on the CPU yields only minor improvement. Therefore, the scheduler prioritizes transferring high-load experts to achieve better overall execution time.

B. Runtime Dynamic Adjustment Mechanism

To support balanced inference under dynamic expert utilization, we introduce a runtime adjustment mechanism.

Step 1: Monitor Total Execution Time Difference. The expected total execution time at layer l for CPU and GPU can be computed in Equation 2.

At the beginning of each layer, the system prioritizes the computation of cold experts on CPU since transferring their weights to GPU would have a poor cost-benefit ratio. When the imbalance of execution time between CPU and GPU exceeds a threshold δ (default 15%, chosen based on experiments: smaller values trigger rebalancing too frequently, larger values reduce its effectiveness):

$$|T_l^{\text{CPU}} - T_l^{\text{GPU}}| > \delta \cdot \max(T_l^{\text{CPU}}, T_l^{\text{GPU}}) \tag{7}$$

the system triggers a dynamic adjustment.

Step 2: Expert Migration Scheme. Let $\check{T}_l^{\text{CPU}}$ and $\check{T}_l^{\text{GPU}}$ denote the remaining execution times on the CPU and GPU, respectively.

(a) CPU slower than GPU: If $\check{T}_l^{\text{CPU}} > \check{T}_l^{\text{GPU}}$, the scheduler transfers a subset of experts from the CPU to the GPU to rebalance the execution time, ensuring that:

$$\check{T}_l^{\text{CPU}} - t_{CPU} = \check{T}_l^{\text{GPU}} + t_{GPU}, \tag{8}$$

$$\check{T}_l^{\text{GPU}} > N_E \times T_{\text{trans}} \tag{9}$$

where t_{CPU} and t_{GPU} denote the CPU and GPU execution times induced by the transferred experts, respectively. N_E is the number of transported experts, and T_{trans} is the expert-weight transfer time.

Based on the performance modeling module described in Section 3.2, we have:

$$t_{CPU} = \alpha \cdot t_{GPU} = \frac{T_{\text{CPU}}(E_i)}{T_{\text{GPU}}(E_i)} \cdot t_{GPU} \tag{10}$$

where α is the relative computational speed between the GPU and CPU, which is stable under same platform and model configurations (see Table 3), e.g., $\alpha = 11.1$ for Mixtral-8×7B as shown in Table 1.

Consequently, t_{CPU} can be obtained and used to guide the scheduler in selecting a subset of unexecuted experts from $\mathcal{C}_{\text{stay}}$:

$$t_{CPU} = \frac{\check{T}_l^{\text{CPU}} - \check{T}_l^{\text{GPU}}}{1 + \alpha} \tag{11}$$

Then the scheduler selects a subset of unexecuted experts from $\mathcal{C}_{\text{stay}}$, sorted by heat in descending order, such that their total expected excution time on CPU is close to t_C, and the number of selected experts satisfies Equation 9. These experts are then transferred to the GPU. The process iterates until the imbalance falls below δ.

(b) GPU slower than CPU: If $\check{T}_l^{\text{CPU}} < \check{T}_l^{\text{GPU}}$, experts are migrated from GPU to CPU according to:

$$\check{T}_l^{\text{CPU}} + t_{CPU} = \check{T}_l^{\text{GPU}} - t_{GPU}, \quad t_{CPU} = \alpha \cdot t_{GPU} \tag{12}$$

where selected experts from $\mathcal{G}_{\text{stay}}$ are reassigned to CPU by heat in descending order. Since CPU memory is sufficient to store all expert weights, only activation tensors are transferred, making this operation highly efficient.

Through this dynamic adjustment process, the system continuously maintains load balance between CPU and GPU, ensuring high hardware utilization and minimizing inference latency.

3.4 Cross-Layer Prefetching and Chunked Transfer Mechanism

Traditional on-demand loading triggers expert weight transfers only when an expert is needed, causing GPU stalls and increased inference latency. As shown in Table 1, weight movement is a major bottleneck, and when transfer latency T_{trans} is large, Equation 9 is rarely satisfied, so dynamic rebalancing is seldom activated.

To overcome this limitation, MoE-Balance introduces cross-layer prefetching and chunked transfer, which predict future expert usage and initiate asynchronous weight transfers ahead of time. This enables deep overlap between computation and communication, hides transfer latency, and significantly improves system throughput.

A. Expert Demand Prediction

The prefetching mechanism predicts the experts required in upcoming layers by leveraging the gate network, which maps input representations to expert selections. Formally, given the hidden states h_l of layer l, the predicted expert demand set for the next layer is computed as [9]:

$$G_{l+1}(h_l) = \text{Softmax}(\text{TopK}(h_l \cdot W^g_{l+1})) \tag{13}$$

where W^g_{l+1} is the gate network weight matrix. While this formula describes single-layer prediction, the method can be extended recursively to deeper layers, allowing early initiation of weight transfers and increasing the available time for large-scale expert movement. This multi-layer forecasting enables the system to initiate transfers at earlier stages, effectively extending the available time window for large-scale expert weight movement.

B. Chunked Transfer Strategy

Although cross-layer prefetching helps overlap communication with computation, the large size of expert weights still makes it difficult to fully hide data transfer latency. To address this limitation, MoE-Balance introduces a chunked transfer mechanism that partitions each expert's weights W into m independent chunks, where

$$W = [W_1, W_2, \ldots, W_m], \qquad T_{\text{chunk}} = \frac{T_{\text{transfer}}(E_i)}{m} \tag{14}$$

each chunk W_j can be transferred independently.

Although chunking does not reduce total communication volume, it enables much finer grain overlap between computation and transfer. As shown in Figure 4, transferring an entire weight tensor forces computation to wait for the full block to arrive, whereas chunked transfer allows computation to begin once the first chunk (W_1) is ready while subsequent chunks are still being transferred. This pipelined execution improves PCIe utilization and hides communication latency, effectively reducing expert loading overhead.

By combining cross-layer prefetching with chunked transfer, the weight transfer latency can be fully hidden by computing. This enables MoE-Balance to increase communication–computation overlap, shorten idle GPU periods, and improve end-to-end inference throughput.

4 Evaluation

4.1 Setup

Platforms We evaluate the performance of MoE-Balance on an NVIDIA GeForce RTX 3090 GPU and an Intel Xeon Gold 6133 CPU. The CPU comprises 2 sockets with 20 physical cores each (40 cores/80 threads in total), operating at 2.50 GHz, and the system is configured with 251 GB of host memory.

Datasets We evaluate our system using the MTBench [19] benchmark. To simulate different generation scenarios, we replicate the dataset to thousands of questions and vary the maximum output token length across 32, 64, 128, and 256 tokens.

Baselines We evaluate our system against five representative MoE inference frameworks: MoE-Infinity [17], AdapMoE [20], HybriMoE [21], and Fiddler [13], which have already been described in Section 2.

Models We evaluate our system on two representative MoE models: Mixtral-8×7B [12] and DeepSeek-V2-Lite-Chat [4], summarized in Table 4. These models reflect two distinct MoE design philosophies, enabling a comprehensive assessment of our system across different expert granularities and activation patterns.

Table 4: MoE Models Configuration.

	Mixtral-8×7B	DeepSeek-V2-Lite-Chat
Layers	32	27
Experts	8	64 (including 2 shared experts)
Top-k	2	6
Expert Params	45.1B	2.4B
Params	46.6B	16B

Metrics We evaluate performance using throughput (tokens/s), computed as the total number of output tokens divided by the sum of prefill and decode time.

4.2 End-to-End Results

We evaluate MoE-Balance using two metrics: throughput (tokens/s) and end-to-end latency. The comparison includes MoE-Infinity, AdapMoE, Fiddler, and HybriMoE across generetion lengths $L \in 32, 64, 128, 256, 512$. Figure 7 reports results under different GPU memory budgets and CPU logical thread counts, both enforced by restricting available system resources.

MoE-Balance consistently outperforms all baselines across all input lengths and hardware configurations. MoE-Infinity uses weight prefetching but cannot fully hide PCIe transfer time, resulting in low throughput, while the 16 GB results are omitted due to **out-of-memory (OOM)** errors. AdapMoE improves communication–computation overlap but does not exploit CPU compute capacity. Fiddler and HybriMoE both utilize CPU computation. However, the static expert placement in Fiddler causes CPU overload when logical threads decrease, which significantly increases latency (16 GB results also omitted due to **OOM**). HybriMoE adapts scheduling but still cannot achieve full overlap, so performance degrades as CPU threads decrease.

MoE-Balance achieves the best results across all experimental settings, with throughput improvements of **33.5**×, **2.3**×, **3.7**×, and **1.6**× over MoE-Infinity, AdapMoE, Fiddler, and HybriMoE, respectively, and end-to-end latency is reduced by **92%**, **68%**, **77%**, and **37.5%** over these aforementioned baseline methods, respectively. Increasing GPU memory from 16 GB to 24 GB yields a modest improvement of throughput of approximately **1.05**× relative to the 16 GB configuration.

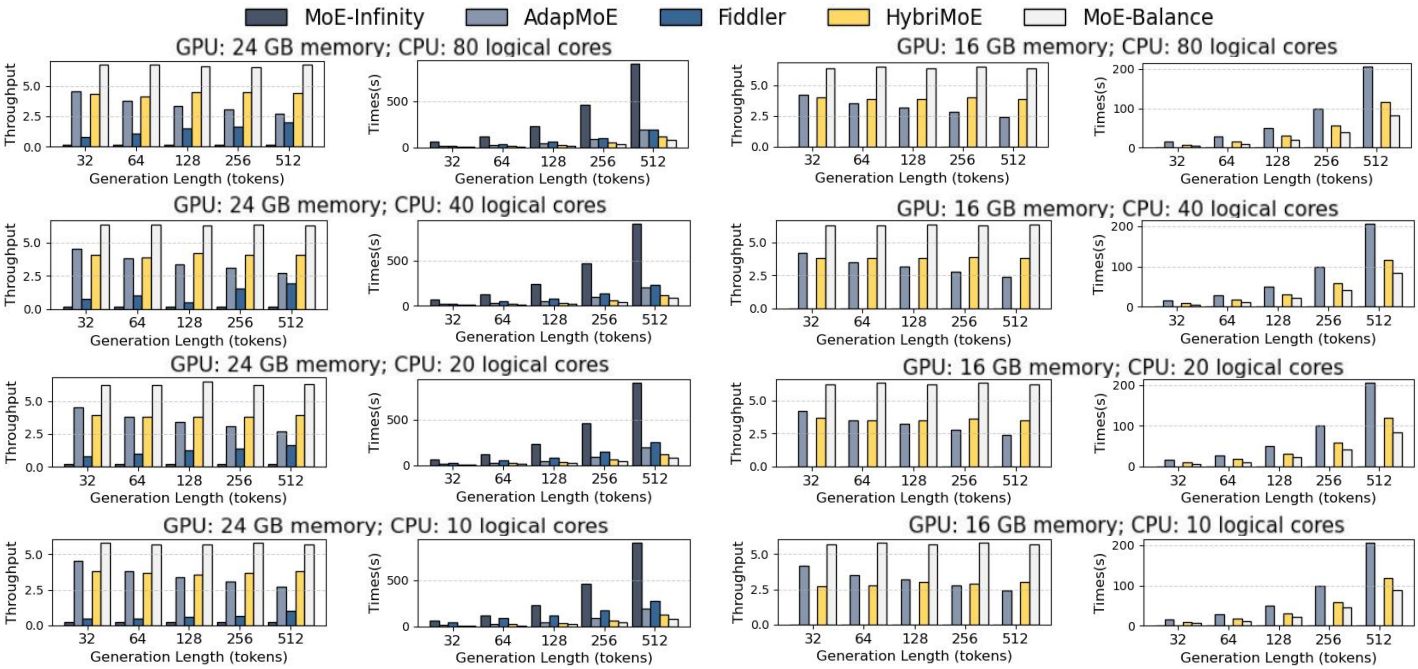


Figure 7: Performance of MoE-Balance vs. Baselines under Different GPU Memory Budgets and CPU Thread Configurations.

4.3 CPU–GPU Execution Balance

We measure the per-layer CPU time, GPU time, and overall MoE layer progression time for all methods, normalized to MoE-Infinity's GPU time. As shown in Figure 8, MoE-Balance achieves the most balanced CPU–GPU usage, effectively reducing pipeline bubbles and improving cross-device overlap.

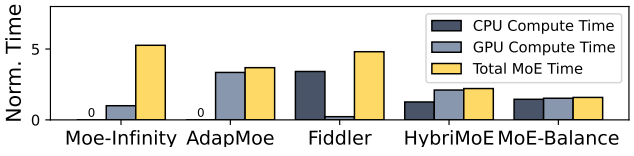


Figure 8: Execution-Time Breakdown.

4.4 Generality of MoE-Balance

To demonstrate generality, we further evaluate it on DeepSeek-V2-Lite-Chat using the configuration in Table 4, and compare against the HybriMoE. Figure 9 reports throughput and end-to-end latency under a 24 GB GPU memory limit and 80 logical CPU cores. On DeepSeek-V2-Lite-Chat, MoE-Balance increases throughput by 1.4× and reduces latency by 29% compared to HybriMoE.

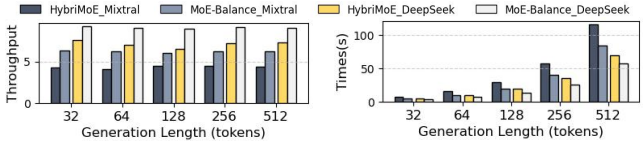


Figure 9: Performance Comparison of MoE-Balance and HybriMoE on Mixtral-8x7B and DeepSeek-V2-Lite-Chat.

4.5 Ablation Study

We analyze the impact of chunk number m and threshold β on end-to-end latency (Figure 10). In this configuration, $m = 3$ and $\beta = 0.2$ achieve the best trade-off: larger m increases scheduling overhead, while too high or too low β reduces the effectiveness of chunked transfers.

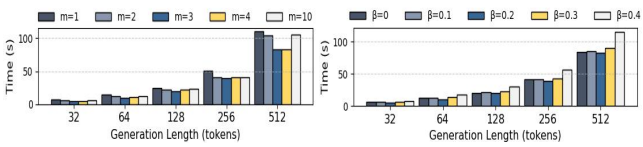


Figure 10: The impact of m and β on end-to-end latency.

5 Conclusion

This paper presents **MoE-Balance**, a heterogeneous CPU–GPU system that maximizes resource utilization and improves MoE inference efficiency. By dynamically scheduling expert tasks and leveraging cross-layer prefetching with chunked transfers, MoE-Balance effectively hides the latency of loading large expert weights. Evaluations on models such as Mixtral-8×7B show that MoE-Balance achieves 1.6× higher throughput and 37.5% lower end-to-end latency compared with state-of-the-art hybrid baselines, highlighting its effectiveness for efficient and scalable MoE inference on resource-constrained platforms.

Acknowledgments

This work was supported in part by the National Key Research and Development Program of China No. 2023YFB4404404, in part by the Key Technologies R&D Program of Jiangsu (Prospective and Key Technologies for Industry) under Grant BE2023005-2, in part by the National Natural Science Foundation of China under Grants 92373205 and 62374146, in part by the key R&D programme of Zhejiang Province No. 2024C01012, in part by CIE-Smartchip research fund No. 2023-004, and in part by the Ascend AI technology stack.

References

[1] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, and et al. 2022. DeepSpeed- Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15. doi:10.1109/SC41404.2022.00051

[2] Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, and et al. 2025. MoE-Lightning: High-Throughput MoE Inference on Memory-constrained GPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 715–730. doi:10.1145/3669940.3707267

[3] Damai Dai, Chengqi Deng, Chenggang Zhao, R.x. Xu, Huazuo Gao, and et al. 2024. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 1280–1297. doi:10.18653/v1/2024.acl-long.70

[4] DeepSeek-AI, Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, and et al. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434 [cs.CL] https://arxiv.org/abs/2405.04434

[5] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Finetuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 10088–10115. https://proceedings.neurips.cc/paper_files/paper/2023/file/1feb87871436031bdc0f2beaa62a049b-Paper-Conference.pdf

[6] Artyom Eliseev and Denis Mazur. 2023. Fast Inference of Mixture-of-Experts Language Models with Offloading. arXiv:2312.17238 [cs.LG] https://arxiv.org/abs/2312.17238

[7] Elias Frantar and Dan Alistarh. 2023. QMoE: Practical Sub-1-Bit Compression of Trillion-Parameter Models. arXiv:2310.16795 [cs.LG] https://arxiv.org/abs/2310.16795

[8] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv:2210.17323 [cs.LG] https://arxiv.org/abs/2210.17323

[9] Siqi He, Haozhe Zhu, Jiapei Zheng, Lizhou Wu, Bo Jiao, and et al. 2025. Hydra: Harnessing Expert Popularity for Efficient Mixture-of-Expert Inference on Chiplet System. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. doi:10.1109/DAC63849.2025.11133023

[10] Ranggi Hwang, Jianyu Wei, Shijie Cao, Changho Hwang, Xiaohu Tang, and et al. 2024. Pre-gated MoE: An Algorithm-System Co-Design for Fast and Scalable Mixture-of-Expert Inference. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 1018–1031. doi:10.1109/ISCA59077.2024.00078

[11] Raj Jain. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley-Interscience.

[12] Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, and et al. 2024. Mixtral of Experts. arXiv:2401.04088 [cs.LG] https://arxiv.org/abs/2401.04088

[13] Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. 2025. Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models. In *The Thirteenth International Conference on Learning Representations*. https://openreview.net/forum?id=N5fVv6PZGz

[14] Saeed Masoudnia and Reza Ebrahimpour. 2014. Mixture of experts: a literature survey. *Artificial Intelligence Review* 42, 2 (2014), 275–293.

[15] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, and et al. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In *International Conference on Learning Representations*. https://openreview.net/forum?id=B1ckMDqlg

[16] Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, and et al. 2024. HOBBIT: A Mixed Precision Expert Offloading System for Fast MoE Inference. arXiv:2411.01433 [cs.LG] https://arxiv.org/abs/2411.01433

[17] Leyang Xue, Yao Fu, Zhan Lu, Luo Mai, and Mahesh Marina. 2025. MoE-Infinity: Efficient MoE Inference on Personal Machines with Sparsity-Aware Expert Cache. arXiv:2401.14361 [cs.LG] https://arxiv.org/abs/2401.14361

[18] Yushu Zhao, Yubin Qin, Yang Wang, Xiaolong Yang, Huiming Han, and et al. 2025. MoBiLE: Efficient Mixture-of-Experts Inference on Consumer GPU with Mixture of Big Little Experts. arXiv:2510.12357 [cs.CL] https://arxiv.org/abs/2510.12357

[19] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, and et al. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 46595–46623. https://proceedings.neurips.cc/paper_files/paper/2023/file/91f18a1287b398d378ef22505bf41832-Paper-Datasets_and_Benchmarks.pdf

[20] Shuzhang Zhong, Ling Liang, Yuan Wang, Runsheng Wang, Ru Huang, and Meng Li. 2024. AdapMoE: Adaptive Sensitivity-based Expert Gating and Management for Efficient MoE Inference. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24)*. ACM, 1–9. doi:10.1145/3676536.3676741

[21] Shuzhang Zhong, Yanfan Sun, Ling Liang, Runsheng Wang, Ru Huang, and Meng Li. 2025. HybriMoE: Hybrid CPU-GPU Scheduling and Cache Management for Efficient MoE Inference. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7. doi:10.1109/DAC63849.2025.11133274

[22] Yuxin Zhou, Zheng Li, Jun Zhang, Jue Wang, Yiping Wang, and et al. 2025. FloE: On-the-Fly MoE Inference on Memory-constrained GPU. arXiv:2505.05950 [cs.LG] https://arxiv.org/abs/2505.05950