MoE-Lens: Towards the Hardware Limit of High-Throughput MoE LLM Serving Under Resource Constraints

Yichao Yuan
University of Michigan
Ann Arbor, Michigan, USA
yichaoy@umich.edu

Lin Ma
University of Michigan
Ann Arbor, Michigan, USA
linmacse@umich.edu

Nishil Talati
University of Michigan
Ann Arbor, Michigan, USA
talatin@umich.edu

Abstract

Mixture of Experts (MoE) LLMs, characterized by their sparse activation patterns, offer a promising approach to scaling language models while avoiding proportionally increasing the inference cost. However, their large parameter sizes present deployment challenges in resource-constrained environments with limited GPU memory capacity, as GPU memory is often insufficient to accommodate the full set of model weights. Consequently, typical deployments rely on CPU-GPU hybrid execution: the GPU handles compute-intensive GEMM operations, while the CPU processes the relatively lightweight attention mechanism. This setup introduces a key challenge: *how to effectively optimize resource utilization across CPU and GPU?* Prior work has designed system optimizations based on performance models with limited scope. Specifically, such models do not capture the complex interactions between hardware properties and system execution mechanisms. Therefore, previous approaches neither identify nor achieve the hardware limit.

This paper presents MoE-Lens, a high-throughput MoE LLM inference system designed through holistic performance modeling for resource-constrained environments. Our performance model thoroughly analyzes various fundamental system components, including CPU memory capacity, GPU compute power, and workload characteristics, to understand the theoretical performance upper bound of MoE inference. Furthermore, it captures the system execution mechanisms, including workload scheduling and the effects of paged KV cache, to identify the key hardware bottlenecks and accurately predict the achievable throughput. Informed by our performance model, MoE-Lens introduces an inference system featuring a resource-aware scheduler for prefill and decode phases, an execution engine that overlaps their computation, a data transfer mechanism for model weights, and an optimized CPU-based attention implementation. Evaluated on diverse MoE models and datasets, MoE-Lens outperforms the state-of-the-art MoE-Lightening by 4.6× on average (up to 25.5×), with our theoretical model predicting performance with an average 94% accuracy.

Keywords

LLM, MoE, resource-constraint env, high-throughput serving

1 Introduction

The emergence of Mixture-of-Experts (MoE) models [11, 16, 37] has marked a significant evolution in the design of Large Language Models (LLMs). In contrast to dense models [5, 20] that activate the full set of model parameters for every input, MoE models introduce sparsity by routing each input through a small subset of expert networks. This design enables MoE models to scale up the total parameter count substantially without proportionally increasing the number of floating-point operations (FLOPs) per inference step. As a result, MoE-based LLMs have demonstrated strong empirical performance across a wide range of tasks [11, 15], while maintaining manageable compute requirements for inference.

However, the practical deployment of MoE models is challenging due to their high memory capacity demand to store the model weights. Although only a few experts are activated per token, all expert parameters must reside in memory to allow flexible routing decisions at runtime. This leads to substantial memory pressure that often exceeds the capacity of a GPU. For instance, recent models such as DeepSeek-V3/R1 [15, 16] and Mixtral-8x22B [2] size hundreds of gigabytes, significantly outpacing the requirements of dense models with similar FLOPs. These memory capacity demands hinder the use of MoE models in *resource-constrained environments*, such as low-cost servers, where the available GPU memory capacity is far less than the model size.

A key technique for enabling MoE inference in resource constrained environments is *CPU offloading* [9, 38]. In this approach, the model weights and Key-Value (KV) cache are stored in CPU memory and transferred to the GPU on demand during inference. As a result, CPU-GPU IO becomes a critical bottleneck. Prior work has sought to mitigate this bottleneck through improved scheduling strategies, including pipelining [38], attention offloading [9, 24], and model-aware prefetching [19]. While these techniques have led to notable, hardware utilization remains low even in state-of-the-art systems, leaving room for significant performance improvements. For example, we find only 16.5% of GPU utilization for MoE-Lightning [9] during the generation stage. This raises important questions: ***what is the upper bound on achievable performance, and how can the system achieve such upper bound?***

In this paper, we present MoE-Lens, a high-throughput MoE inference framework designed for resource-constrained environments, achieving up to 25.5× and an average of 4.6× speedup over state-of-the-art MoE-Lightening [9]. MoE-Lens's design contains three stages. First, unlike prior work that relies on limited-scope performance models for system optimizations, MoE-Lens employs a two-stage ***holistic performance model*** that considers factors beyond CPU-GPU I/O bandwidth. It not only identifies theoretical performance upper bounds but also accurately predicts the execution time. In the third stage, we propose a system design guided by this model and jointly optimize the execution pipeline and sequence-level scheduling to bring the system closer to hardware limits. Similar to prior works [9, 19, 38, 45], our focus is *offline, batching processing* inference tasks, such as model evaluation [27],

Conference'17, July 2017, Washington, DC, USA
2025. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

data wrangling [31], form processing [10], LLM for relational analytics [30], and synthetic data generation [20], where maximizing inference throughput directly reduces total job completion time.

MoE-Lens's two-stage performance modeling accounts for critical factors that represent missed opportunities in prior works [9, 19, 38] to show how they influence throughput. In the first stage, the model analyzes the theoretical performance upper bound of MoE inference based on the fundamental system components. It identifies *CPU memory capacity*, an element overlooked by prior work, as a primary limiting factor and quantifies how prompt and generation lengths impact the memory utilization. The second stage captures how system execution mechanisms, including workload scheduling and paged KV cache, affect memory/compute utilization and overall system performance. By integrating all these dimensions, our model accurately predicts end-to-end wall-clock inference time for systems operating near hardware limits.

In the third stage, MoE-Lens introduces a high-throughput MoE LLM inference system design informed by our holistic performance model that significantly outperforms state-of-the-art solutions. The system maximizes hardware utilization by addressing key inefficiencies in CPU-side resource usage and balancing compute across the prefill and decode stages. To this end, we introduce resource-aware scheduling that enables effective prefill/decode overlapping, reducing idle time and smoothing workload distribution. We also propose a novel pipeline design, VSLpipe, which includes a contiguous data mover to maximize CPU-GPU bandwidth utilization during weight transfers. Our hand-optimized CPU attention kernel using instrinsics fully leverages the vector units of modern CPUs, preventing the CPU compute throughput from becoming a bottleneck and improving GPU utilization.

Evaluated on diverse models and datasets, MoE-Lens achieves on average 4.6×, up to 25.5×, speedup over the state-of-the-art solution MoE-Lightning [9]. The results show the importance of holistic performance modeling and architecture-aware design decisions for high-throughput MoE inference in resource-constrained environments. In summary, MoE-Lens makes the following contributions.

- A holistic performance model for MoE inference in resource-constrained environments capturing complex interactions between hardware properties and system execution mechanisms.
- An accurate throughput predictor for MoE LLM inference under hardware constraints.
- A system design informed by the model, featuring resource-aware phase scheduling, CPU-side attention execution, and efficient weight/KV cache transfer.
- MoE-Lens: an architecture-aware, end-to-end CPU-GPU hybrid system with theoretical underpinnings that achieves an average 4.6× throughput improvement over the state-of-the-art.

2 Background

Mixture-of-Expert (MoE) LLMs. MoE LLMs achieve strong benchmark performance while reducing compute needs compared to dense LLMs with similar parameter counts, primarily composed of attention and MoE layers. For architectural details, see [9, 19]. A defining trait of modern MoE models is their large size: hundreds of GBs [14–16]—which exceeds standard GPU memory [9, 19, 38].

Concepts in LLM Model Inference. One key module in LLM Models is the attention module, where *key* and *value* vectors are calculated and cached in *KV Cache*. *Group Query Attention (GQA)* is a commonly used attention variant in MoE models, which allows a group of query vectors to share a single pair of key and value vectors, thereby reducing the size of the KV cache. The LLM inference consists of two stages: the *prefill stage*, typically *compute-bound*, where the initial prompt is processed in parallel, and the *decode stage*, typically *memory-bound* [33], where tokens are generated sequentially in an auto-regressive manner.

Resource-Constrained LLM Inference. LLM inference in resource-constrained environments prioritizes high-throughput batch processing on systems where the GPU memory is significantly smaller than the model's total parameter size, while the CPU has sufficient memory or disk capacity to store model weights. This setting differs from traditional LLM serving systems [26, 33, 36] optimized for latency-sensitive applications like chatbots and code completion, where low response time is critical. Instead, resource-constrained inference systems, typically equipped with GPUs like T4 (16GB) or L4 (24GB) [9, 19, 38], prioritize overall throughput and can afford to trade off latency. Since these GPUs lack sufficient memory to store the full model, weights must be streamed from CPU memory over PCIe, introducing substantial overhead. MoE-Lightning [9] addresses this by offloading attention computation to the CPU, forming a CPU–GPU hybrid system. This design avoids transferring the large KV cache to GPU, which is essential as growing GPU parallelism leads to KV sizes exceeding memory capacity. Moreover, because attention has low arithmetic intensity [9, 13], the CPU can execute it efficiently while the GPU focuses on compute-intensive MoE layers.

3 Motivation

While MoE-Lightning [9], the state-of-the-art MoE inference system for resource-constrained environments, leverages a performance model to guide system design and achieves substantial speedups, an important question remains: ***does the state-of-the-art fully harness the capabilities of the underlying hardware?***

3.1 Limited-Scope of Performance Modeling in Prior Work

MoE-Lightning introduced the Hierarchical Roofline Model (HRM) to address the CPU-GPU IO bottleneck, achieving notable gains by offloading decode-stage attention to the CPU and avoiding frequent KV cache transfers. While effective, HRM's focus is limited to arithmetic intensity and IO bandwidth, overlooking two crucial factors that influence performance ceilings: CPU memory capacity and the characteristics of input requests, such as prompt and generation lengths. These factors directly affect how much weight transfer overhead can be amortized and how well pipelining can hide data movement latency. As shown in Table 1, typical execution plans generated by MoE-Lightning result in underutilized CPU memory, revealing inefficiencies in resource allocation. Moreover, sustaining high concurrency requires not just fast IO, but also ample memory bandwidth and compute throughput on the CPU side: elements

Prefill Length	Generation Length	CPU Memory (GB)	CPU Memory Utilization
98	32	265	52.0%
98	64	265	56.2%
926	128	265	35.0%

Table 1: CPU memory utilization for execution plans generated by MoE-Lightening [9], showcasing under-utilization.

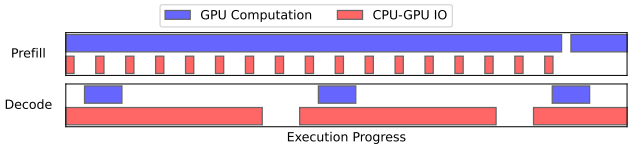


Figure 1: Sample of an execution timeline of GPU computation and CPU-GPU IO during the prefill and decode stages of MoE-Lightning.

HRM does not model. This presents a *significant opportunity* to rethink scheduling and architecture-aware execution strategies that better align with hardware constraints.

Motivated Approach. These limitations motivate the development of the *Stage 1 Model* in MoE-Lens's three-step approach, which extends beyond operator-level IO analysis to capture a more complete picture of system performance. This model establishes a theoretical upper bound on throughput for processing requests with varying prompt and generation lengths, incorporating the often-overlooked impact of CPU memory capacity.

3.2 Resource Utilization Imbalance between Prefill and Decode Stages.

To approach the throughput upper bound, it is essential to account for *workload heterogeneity* in LLM inference, which causes imbalanced resource utilization between the prefill and decode stages. As shown in Figure 1, the prefill stage fully utilizes GPU compute but leaves CPU-GPU IO bandwidth underutilized, while the decode stage suffers from low GPU utilization due to on-demand weight transfers from CPU to GPU, constrained by limited GPU memory. Existing MoE inference systems in resource-constrained settings handle these stages separately, simplifying scheduling but exacerbating the imbalance. Profiling MoE-Lightning [9] with a 98-token prompt and 32-token generation reveals this inefficiency: CPU-GPU IO is active only 23.9% of the time during prefill, while GPU utilization drops to 16.5% during decode.

Motivated Approach. This observation motivates the need to *overlap prefill and decode execution* to approach the hardware limit. MoE-Lens formalizes this in the *Stage 2 Model*, which incorporates key execution factors to make the model both resource- and workload-aware while still reflecting hardware constraints. By accounting for overlapped scheduling and system-level interactions, the *Stage 2 Model* aligns closely with real system behavior, enabling practical guidance for system design and accurate estimation of end-to-end execution time in MoE inference.

4 Overview of MoE-Lens

In this work, we aim to answer the following critical questions:

- *What is the throughput upper bound for a machine?*
- *How to systematically approach that performance upper bound?*

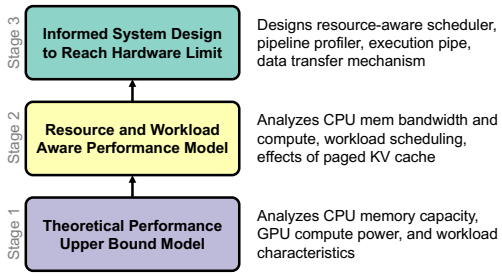


Figure 2: Overview of MoE-Lens that combines a theoretical performance upper bound, resource and workload aware performance model, and an informed system design to reach hardware limits.

In MoE-Lens, we take a three-step approach as shown in Figure 2, building up an increasingly detailed understanding of MoE inference in a resource-constrained environment, and eventually reach a concrete design that reaches the hardware limit. In *Stage 1*, we develop a model to understand the theoretical performance upper bound of MoE inference based on the fundamental system's architectural components (§5.1, §5.2). We further analyze the requirements on CPU memory bandwidth and compute throughput, which support the performance upper bound (§5.3) and the benefits of prefill/decode overlapped scheduling (§5.4). Holistically considering all factors above, and request batch size and paged KV cache, we derive our resource and workload aware performance model in *Stage 2* from an implementation viewpoint (§5.5), providing realistic insights for concert design and predict system performance (with 94% accuracy). One important property of our *Stage 2* model is that it *converges* to the performance upper bound with an increasing batch size, thus still modeling the hardware limits but pricing in the physical execution factors. Finally in *Stage 3*, we provide a system design based on the insights from modeling and the guidance of the *Stage 2* model, which adapts to the variance of the real execution environments while achieving the high hardware utilization outlined by the *Stage 2* model (§6).

5 MoE-Lens Performance Model

This section describes the details of our *Stage 1* and *2* models.

5.1 CPU Memory Capacity as a Limiting Factor

As the CPU memory stores the KV cache, its capacity directly determines the number of sequences that can be processed in parallel. A large number of active sequences allows a large number of tokens to reuse the weights loaded from the CPU for computation, amortizing the cost of moving the weights from the CPU to the GPU. *An important question is* ***how much CPU memory is necessary to fully utilize the GPU?***

Let N_e denote the number of experts, N_k the number of top-k experts selected per token token, h the model dimension, h_i the intermediate dimension of the expert networks (typically, $h_i = mh$, where $m > 1$), s the GQA group size and n as the number of tokens processed in parallel. The GEMM arithmetic-to-IO intensity, which is the amount of GEMM computation divided by the amount of

	Sequence Length 256			Sequence Length 512		
NVIDIA GPU	A40	L40	A100	A40	L40	A100
BF16 Throughput (in T FLOPS)	150	181	312	150	181	312
# of Tokens to Saturate GPU compute	19.2k	23.2k	40.0k	19.2k	23.2k	40.0k
KVCache Size to Saturate (in GB)	614	741	1277	1228	1482	2554

Table 2: KV Cache Size Needed to Saturate GPU compute.

weight data accessed, of a MoE model is

$$I = n\frac{6N_k hh_i + 4h^2 + 4\frac{h^2}{s}}{6N_e hh_i + 4h^2 + 4\frac{h^2}{s}} = n\frac{6mN_k + 2 + \frac{2}{s}}{6mN_e + 2 + \frac{2}{s}} \approx n\frac{N_k}{N_e} \tag{1}$$

Here, $\frac{N_k}{N_e}$ reflects the sparsity of the MoE layer. Let C_{GPU} be the GPU's GEMM throughput and B_{IO} the GPU-CPU IO bandwidth. To saturate the GPU computation power, we require

$$I \geq \frac{C_{GPU}}{B} \Leftrightarrow n \geq \frac{C_{GPU}}{B}\frac{N_e}{N_k} \tag{2}$$

For example, saturating the compute of a single NVIDIA A40 GPU (C_{GPU} = 150 TFLOPS, $B = 32$, $N_e = 8$, $N_k = 2$) when running `Mixtral-8x7B` requires processing *19,200 tokens* in parallel. Table 2 quantifies the number of tokens and corresponding KV cache size, assuming a 512-token sequence length (sum of prompt and generation length), needed to saturate different GPUs. Even under a throughput-optimized system design, where abundant pending sequences are assumed, achieving this level of parallelism is challenging due to the **large CPU memory capacity** required to hold the KV cache. Each token contributes to the cumulative KV cache footprint, which can quickly exceed the available CPU memory in resource-constrained settings. For instance, supporting 512-token sequences would require *1.2TB* of CPU memory for KV cache: disproportionate for a single GPU. This memory demand grows with increasing GPU compute capabilities, as illustrated in Table 2.

> ***Takeaway:*** *CPU memory capacity for KV cache storage is a limiting factor to fully utilize GPU compute resources.*

5.2 Stage 1 Model: Parallelism-Memory Efficiency Analysis

While CPU memory capacity constrains the number of tokens that can be processed in parallel, the memory footprint per token varies across stages of inference. During the prefill stage, all tokens in the prompt can be processed simultaneously, effectively amortizing the memory cost across multiple token computations. In contrast, the decode stage operates autoregressively: each sequence's KV cache enables computing only a single token at a time. Consequently, prefill tokens offer higher memory efficiency compared to decode tokens, in terms of parallel computation per unit of memory. This leads to: ***what is the theoretical upper bound on system throughput for a batch of requests with varying prompt and generation lengths, under a fixed hardware configuration?***

We introduce *Parallelism-Memory Efficiency* (*PME*), a metric quantifying how effectively a sequence translates memory capacity into the number of tokens that can be processed in parallel to saturate GPU resources. The *PME* for a sequence s with prompt length p and generation length g is

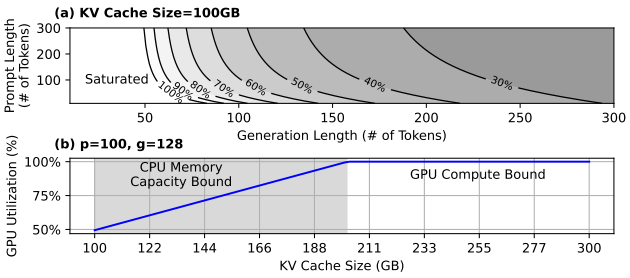


Figure 3: Visualization of the maximum GPU utilization $\frac{T_{max}}{T_{GPU}}$. (a) Maximum GPU utilization when running `Mixtral8x7B` on `A40` with 100GB KV cache. (b) For the same model and GPU, the maximum GPU utilization when $p = 100$ and $g = 128$.

$$\begin{aligned}\text{PME} &= \frac{\sum_{\text{gen. steps}} \text{Parallel Tokens}}{\sum_{\text{gen. steps}} \text{Sequence KV Cache Size}} \\ &= \frac{p+g}{\sum_{j=0}^{g}(p+j)} = \frac{2(p+g)}{(2p+g)g}\end{aligned} \tag{3}$$

Here, the denominator is the sum of the memory capacity a sequence occupies across its entire generation lifetime. We further define the time to transfer the weight of a model from CPU to GPU as $\delta = \frac{\text{Model Size}}{B_{IO}}$. For a batch of sequences with average prefill length p and generation length g, its theoretical maximum inference throughput (tokens/sec) can be estimated as:

$$T_{max} = min(\frac{\text{PME} \cdot M}{\delta}, T_{GPU}), \tag{4}$$

where M is the size of KV cache in number of tokens, and T_{GPU} is the maximum throughput of a GPU in number of tokens per second. Figure 3(a) illustrates how the theoretical maximum GPU utilization varies with prompt length (p) and generation length (g) under a 100GB KV cache budget. Longer sequences lead to lower theoretical GPU utilization, while a higher prompt-to-generation ratio improves utilization for a given sequence length. Figure 3(b) presents a roofline model of theoretical GPU utilization. As KV cache capacity increases, the system transitions from a CPU memory capacity-bound regime to a GPU-bound regime. In the memory-bound regime, limited CPU memory constrains the number of parallel sequences, leading to underutilized GPU compute and throughput that scales with available KV cache. Once the system becomes GPU-bound, the GPU is fully saturated, and further increases in KV cache capacity yield diminishing returns in performance.

> ***Takeaway:*** *Prompt and generation length jointly determine the theoretical upper bound on achievable GPU utilization.*

5.3 CPU Memory Bandwidth and Compute Throughput Requirements

In a hybrid CPU-GPU MoE inference system, the CPU hosts the KV cache and computes decode attention. To avoid bottlenecking the overall execution, the CPU needs to process the attention at a certain throughput, which stresses its memory bandwidth and

compute throughput. *It is crucial to understand:* ***what is the desired CPU memory bandwidth and compute throughput****?*

In a typical deployment scenario, CPU attention computation, CPU-GPU weight transfer, and GPU computation are fully overlapped in the execution pipeline. During each LLM inference iteration, both the model weights and the KV cache are read once from CPU memory, either for transfer to the GPU or for use by the CPU cores. As a result, the total CPU memory bandwidth requirement is the sum of the bandwidth needed to access the KV cache (B_{KV}) and the bandwidth to transfer weights from CPU to GPU (B_{IO}).

$$B_{Mem} = B_{KV} + B_{IO} = \frac{M}{M_{weight}/B_{IO}} = \frac{M}{M_{weight}} B_{IO} \tag{5}$$

Here, M and M_{weight} are the total memory capacity and the memory capacity reserved for the model weight, respectively. The required CPU compute throughput for attention is related to the memory access bandwidth for the KV cache. Assuming that the CPU up-converts the KV cached stored in BF16 to FP32 for computation, we have

$$T_{CPU} = 2 \cdot s \cdot I_{\text{cpu\_attn}} \cdot B_{KV} \tag{6}$$

where s is the GQA group size, and $I_{\text{cpu\_attn}}$ is the arithmetic intensity of attending to one query in the KV cache. $I_{\text{cpu\_attn}}$ is inherently small for the flash attention based implementation, as its primary operations are vector doc product and `saxpby`.

In resource-constrained environments, the CPU memory capacity is usually limited. While the CPU memory bandwidth requirement increases with the size of the KV cache, it typically does not become a bottleneck on modern CPUs equipped with multiple memory channels: often supporting bandwidths exceeding 100GB/s. For instance, consider an extreme case where the KV cache is twice the size of the model weights. In the case of `Mixtral8x7B`, this corresponds to a 200GB KV cache. According to Equation (5), the memory bandwidth required to avoid becoming a performance bottleneck is approximately three times the PCIe bandwidth, *i.e.,* $B_{\text{Mem}} = 60\,\text{GB/s}$. This is well within the capabilities of modern CPUs. On the other hand, sustaining GPU utilization requires the CPU attention computation to deliver throughput on the order of hundreds of GFLOPs. Achieving this performance necessitates efficient use of vector units on modern CPUs. We provide a detailed analysis of this challenge and our implementation in §6.6.

> ***Takeaway:*** *Fully utilizing CPU vector units and memory bandwidth through multiple channels is essential for high-throughput MoE inference.*

5.4 Understanding the Effect of Prefill - Decode Overlap

In §3.2, we briefly explain the motivation behind overlapping the prefill and decode stages. *Another important benefit of this strategy is:* ***prefill-decode overlap effectively improves the CPU memory usage.*** Overlapping allows the decoding process of some sequences to start earlier than others. This reduces the peak memory consumption of a batch of sequences in KV cache, as at each forward pass, some sequences finish execution and release their memory in the KV cache. For an inference batch with on average p prefill tokens and g generation tokens, when using prefill/decode overlapping, we effectively enlarge the KV cache capacity, C_{KV}, to

$$C_{KV,eff.} = \frac{p+g}{p+\frac{g}{2}} C_{KV} \tag{7}$$

compared to making the prefill/decode two isolated stages, without severely overflowing the KV cache. The memory usage per sequence can be estimated using the average sequence length $p + g/2$ in the KV cache instead of the maximum sequence length $p + g$. It is critical for resource-constrained MoE inference, where the memory capacity is a key limiter for large execution batch size.

> ***Takeaway:*** *Besides balancing hardware resource usage, prefill/decode overlapping effectively enlarges the KV cache.*

5.5 Stage 2 Model: Modeling Performance in Realistic Setting

In a realistic deployment scenario, the number of sequences in a batch request is not unbounded, but limited to a large finite value K. Additionally, the KV cache is typically organized into blocks [26], each containing b token slots, with a total of N blocks. In this section, we present our resource- and workload-aware performance model (*Stage 2* in §4). This model builds upon the idealized analysis in *Stage 1*, while incorporating additional considerations such as CPU resource constraints, prefill/decode overlapped scheduling strategies, bounded request batch size K, and paged KV cache organization. The goal is to provide a performance model that accurately predicts end-to-end execution time and remains practical for guiding real system designs (*Stage 3*), while still converging to the hardware-limited upper bound established in *Stage 1.*

We define $\delta = \frac{\text{Model Size}}{B_{IO}}$ same as §5.2. When the number of tokens processed in parallel is less than the GPU's limit T_{GPU}, each inference iteration takes time δ to finish as IO becomes a bottleneck. In each iteration, we schedule q sequences for prefill, where q is calculated as

$$q = \frac{N}{\sum_{i=0}^{g} \lceil \frac{p+i}{b} \rceil} \tag{8}$$

Such a scheduling strategy ensures maximal utilization of the KV cache without over-committing it. In each inference iteration, $q(p+g)$ tokens are released and generated, due to the completion of older sequences and the progression of generation. In contrast, under a naive scheduling approach that strictly separates the prefill and decode stages, the number of parallel decode tokens is limited to $\frac{N}{p+g}$. More active sequences can be decoded in parallel as

$$gq = g\frac{N}{\sum_{i=0}^{g} \lceil \frac{p+i}{b} \rceil} > g\frac{N}{g(p+g)} = \frac{N}{p+g} \tag{9}$$

The generation throughput for a batch with K sequences is

$$T_1 = \frac{K \cdot g}{(\frac{K}{q} + g)\delta} = \frac{K}{K+gq}\frac{gq}{\delta} \tag{10}$$

where $\frac{K}{K+gq}$ is the slowdown factor due to the epilogue of pipelining and $\frac{gq}{\delta}$ is the ideal decoding throughput with gq as the number of sequences active for decoding.

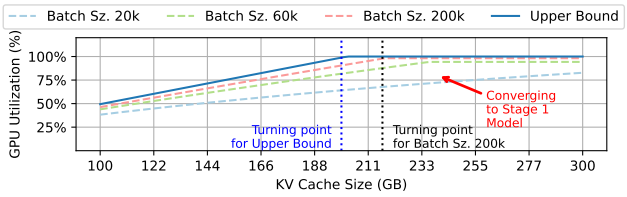


Figure 4: Predicted GPU utilization under different request batch sizes, with $p = 100$ and $g = 128$.

When $T_{GPU} < q(p+g)$, the inference is bottlenecked by GPU computation instead of CPU memory capacity. The total execution time is dominated by the speed we prefill sequences. During execution, both the decode tokens and prefill tokens consume GPU computation. The prefill throughput in the middle of the software pipeline is

$$T_{prefill} = T_{GPU} \frac{p}{p+g} \tag{11}$$

The prologue of the software pipeline takes g iterations, where the average prefill throughput is $\frac{T_{prefill}+T_{GPU}}{2}$. Consequently, $K \cdot p - \frac{T_{prefill}+T_{GPU}}{2} g$ tokens are processed in the main pipeline stages. The total number of iterations for the software pipeline is

$$It = 2 * g + (K \cdot p - \frac{T_{prefill} + T_{GPU}}{2} g)/T_{prefill} \tag{12}$$

The end-to-end throughput is

$$T_2 = \frac{k \cdot g}{It \cdot \delta} \tag{13}$$

An inference process is either bounded by computation or memory capacity, so the throughput will be

$$T = min(T_1, T_2) \tag{14}$$

As shown in §8.1, this model achieves on average 94% accuracy against real execution time.

Impact of real system execution factors. We present the predicted GPU utilization across different request batch sizes in Figure 4. Increasing the batch size improves both the throughput gains from a larger KV cache and the maximum achievable throughput as pipeline prologue and epilogue overhead is amortized. Comparing the curve for a batch size of 200k against the theoretical upper bound, we observe that the use of paged KV cache shifts the turning point to the right, effectively increasing the memory capacity requirement to reach the same level of GPU utilization. *As the batch size theoretically approaches infinity and the KV cache block size approaches 1, the Stage 2 model converges to the Stage 1 theoretical upper bound, delivering the same result.*

> ***Takeaway:*** *Limited number of sequences in a request batch and paged KV cache reduces the effectiveness of a large KV cache size compared to the theoretical analysis.*

6 MoE-Lens System Design

Building on the insights from our architecture-aware performance model (§5), MoE-Lens introduces a system design that closely tracks the theoretical performance limit projected by the *Stage 2* model while remaining adaptive to real-world variations in input sequence lengths and hardware performance.

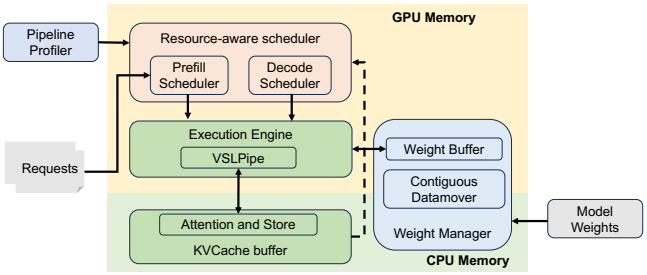


Figure 5: System overview of MoE-Lens.

6.1 Design Overview

Figure 5 illustrates the core components of the MoE-Lens system. When deployed on a given machine for a specific model, the **Pipeline Profiler** identifies the number of tokens required to saturate the GPU compute, guiding the scheduler to prevent pipeline overcommitment. Incoming inference requests are handled by the **Resource-Aware Scheduler**, which overlaps prefill and decode stages and makes scheduling decisions based on the current availability of CPU and GPU resources. At the heart of the execution engine is MoE-Lens's **VSLPipe**—short for *Versatile Pipeline*—which efficiently co-processes prefill and decode requests in resource-constrained environments. It utilizes weight tensors stored in the *Weight Buffer*, managed by the *Weight Manager*, which coordinates with the **Contiguous Data Mover** to eliminate IO stalls during weight transfers. All attention computations are offloaded to the CPU, and our optimized CPU-based flash attention kernel ensures this stage does not become a throughput bottleneck.

6.2 Resource-Aware Scheduler

The goal of the Resource-Aware Scheduler is to schedule sequence execution in a way that (a) fully utilizes available CPU memory capacity, and (b) enables effective prefill/decode overlap, as the models in §5.2 and §5.5 have shown the importance of CPU memory capacity and scheduling strategy in saturing the hardware. Unlike analytical models, the system implementation must adapt dynamically to variations in sequence lengths and hardware performance. The scheduler consists of two components: the **Prefill Scheduler** and the **Decode Scheduler**. Together, they orchestrate the flow of sequences through the prefill and decode stages, ensuring efficient use of memory for KV caching under resource constraints. New sequences are initially queued in the Prefill Scheduler, awaiting prompt processing. In the steady state, there are enough prefill and decode sequences to process. Once a sequence completes the prefill stage, control is handed off to the Decode Scheduler for token generation. During execution, both schedulers may issue sequences, whether in the prefill or decode stage, for parallel processing. To maximize efficiency, both schedulers run on the GPU and maintain their scheduling states directly in GPU memory.

Figure 6 illustrates the two modes of interaction between the Prefill Scheduler and the Decode Scheduler. Based on the availability of KV cache blocks, the system dynamically switches between: *Normal Inference Mode*, where both schedulers operate concurrently

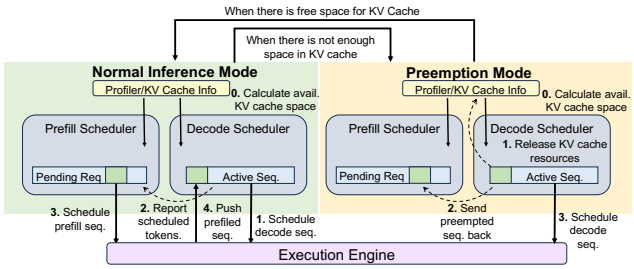


Figure 6: Operation of the Resource-Aware Scheduler.

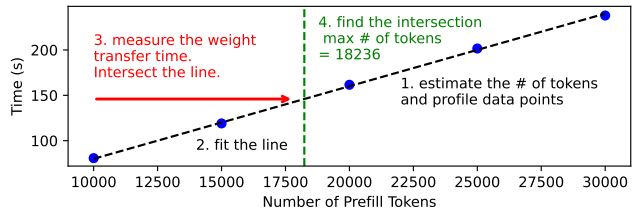


Figure 7: An Example for the threshold n_{real} searching process of MoE-Lens's pipeline profiler, which runs `Mixtral8x7B` on `A40`.

without interference, and *Preemption Mode*, where old decode sequences are prioritized by temporarily preempting new decode sequences to free up memory.

Normal Inference Mode. Before scheduling begins, the Decode Scheduler estimates the number of KV cache blocks required to decode the next token for its managed sequences. If a sufficient number of KV blocks are available, the system enters normal inference mode. In this mode, both the Prefill Scheduler and Decode Scheduler can schedule sequences concurrently. The Decode Scheduler first schedules all sequences currently in the decode stage onto the execution engine. Next, the Prefill Scheduler reads the number of active decode sequences and calculates how many additional prefill tokens it can schedule without exceeding the pipeline capacity, as measured by the Pipeline Profiler (§6.3). It then schedules prefill requests from the head of its queue, staying within this threshold. After the inference pass, newly prefilled sequences are handed off to the Decode Scheduler for subsequent token generation. As decode completes, the Decode Scheduler performs garbage collection to reclaim resources such as KV cache blocks.

Preemption Mode. If the Decode Scheduler detects insufficient KV cache blocks to schedule all decode-stage sequences, the system enters preemption mode. In this mode, a subset of active decode sequences are preempted to free up KV cache resources for other sequences that are ready to generate the next token. The KV cache entries and other associated resources from the preempted sequences are reclaimed and reassigned to the remaining decode sequences. Meanwhile, the Prefill Scheduler halts the scheduling of any new incoming sequences. Instead, it accommodates the preempted decode sequences, treating them as newly arrived sequences that must go through the prefill stage again. This effectively re-inserts them into the execution pipeline from the beginning, but with the advantage that their earlier progress has already been partially completed.

Discusstion. A key design in our scheduler is to alleviate preemption overhead with overlap. When a sequence is preempted, its KV cache is evicted and must be re-prefilled. Without overlapping prefill and decode, this recomputation can only begin after the current decode stage completes, incurring additional latency. In contrast, overlapping prefill with decode allows the recomputation to proceed concurrently, effectively hiding its cost and minimizing impact on overall execution time.

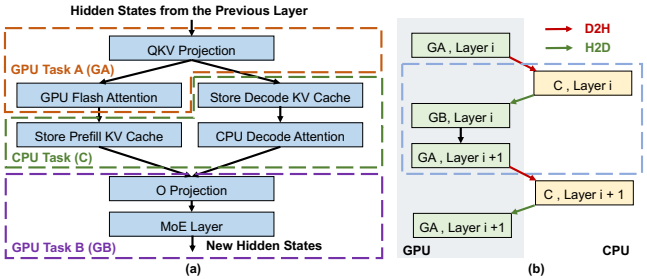


Figure 8: Compute graph division of MoE-Lens's VSLPipe.

6.3 Pipeline Profiler

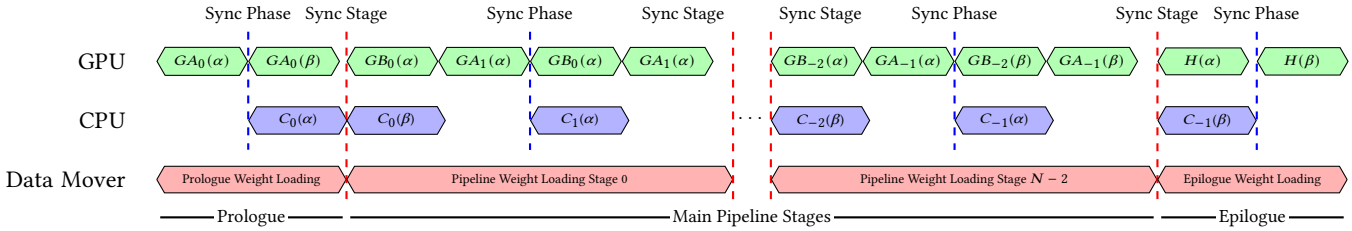
To maintain effective prefill/decode overlap as modeled in §5.5, MoE-Lens uses a Pipeline Profiler to estimate the token threshold n_{real} at which GPU-side GEMM becomes the bottleneck, *i.e.,* the measured n in Equation 2. The resource-aware scheduler ensures that the total number of scheduled tokens stays below n_{real}. This avoids prematurely exhausting prefill sequences and dimensing the effects of prefill/decoce overlapping, thereby sustaining high GPU utilization. The pipeline profiler estimates n using Equation 2, then varies the number of prefilled tokens and measures the corresponding GPU computation time. As shown by an example for `Mixtral8x7B` on the A40 in Figure 7, the profiler fits a line to capture the relationship between token count and GPU time. It also measures the time required to transfer a layer of weights to the GPU. It calculates the maximum number of parallel tokens using the line's slope and weight transfer time.

6.4 Execution Engine

To achieve prefill/decode overlapping without compromising CPU-GPU IO efficiency, as outlined in the performance model (§5.5), MoE-Lens's execution engine features a novel CPU-GPU hybrid pipelined schedule, *VSLPipe*. It is designed to maximize throughput when executing a batch of requests with both prefill and decode sequences.

VSLPipe Compute Graph Division. Figure 8(a) shows how VSLPipe restructures the compute graph of a MoE transformer layer. The per-token *QKV projection* and *GPU Flash Attention* are grouped into *GPU Task A (GA)*, producing KV values for both prefill (post-attention) and decode (pre-attention) sequences. These KV values are offloaded to the CPU, where *CPU Task (C)* stores them in the KV cache and performs *Decode Attention* for decode tokens. The attention results for decode tokens are transferred back to the GPU and combined with prefill outputs. Finally, *GPU Task B (GB)*, which includes the *O projection* and MoE layer, is applied to all tokens.

The CPU and GPU tasks from different layers can be regrouped to better delineate the boundaries between CPU and GPU computation, as illustrated in Figure 8(b). Specifically, the CPU task from layer i, GPU Task B from layer i, and GPU Task A from layer $i+1$ are combined into a single *execution stage*. Each stage consists of



GA_x, GB_x, and C_x are the GPU Task A, GPU Task B, and CPU Task, defined in Figure 8, at layer x. H means GB of the last layer and the model head.

Figure 9: MoE-Lens's VSLPipe execution timeline.

two *execution phases*: a CPU-only phase followed by a GPU-only phase. Between these phases, either a *Device-to-Host (D2H)* or *Host-to-Device (H2D)* transfer occurs: KV values are offloaded to the CPU after GPU computation, and decode attention results are later loaded back to the GPU. Importantly, the amount of computation-related data transferred is relatively small compared to the cost of weight transfers. The total data transferred per stage is bounded by $2n(d + \frac{2d}{s})$, which represents only a small fraction of the overhead when the number of tokens is large enough to saturate the GPU.

VSLPipe Scheduling. Upon receiving prefill and decode jobs, the execution engine partitions them into two groups, α and β, balancing the number of decode and prefill tokens in each. To maximize CPU-GPU utilization, *VSLPipe* applies a software-pipelined execution strategy, as shown in Figure 9. The pipeline comprises a *prologue*, $N-1$ *main stages*, and an *epilogue* for a model with N layers. Each stage is divided into two phases. In each phase, CPU-side attention computations for one partition run concurrently with GPU-side GEMM operations for the other. At the end of each phase, the CPU and GPU synchronize to exchange intermediate results: attended query vectors are transferred from the CPU to the GPU, while fresh query vectors and KV cache entries are offloaded from the GPU to CPU.

The computation-related data transfer is relatively small compared to the overhead of weight transfers. The total data transferred per computation is bounded by $2n(d + \frac{2d}{s})$. For instance, with $n = 18500$, this amounts to roughly 200MB: only a small fraction of the per-layer weight size. To mitigate the CPU-GPU IO bottleneck, weights for the next execution stage are prefetched at the beginning of each stage by the *Contiguous Data Mover*. This data mover runs asynchronously with the computation threads and synchronizes only at the stage boundaries: not at the end of each individual phase. This design allows it to schedule transfers independently and fully utilize the available CPU-GPU bandwidth. We describe the data mover's design in detail in the next section.

6.5 Weight Layout, Weight Buffer, and Contiguous Data Mover

MoE-Lens treats the weights of each layer as two components: layer-wise weights, which include the attention projection matrices and normalization parameters, and expert weights specific to the MoE layers. All weights are stored in pinned CPU memory to enable efficient CPU-GPU IO transfers. During inference, weights are dynamically loaded into the GPU's weight buffer on demand. The size of the weight buffer is two times the model weight size divided by the number of layers. Given that a MoE model usually has tens of layers, the weight buffer is only a few percent of the original model size.

Contiguous Data Mover. Rather than embedding data movement API calls within the execution pipeline, MoE-Lens implements a dedicated Contiguous Data Mover, running on a separate thread, to handle CPU-GPU IO. The execution pipeline pushes weight transfer requests to the data mover at *layer-wise granularity*, while the data mover internally performs fine-grained transfers for efficiency. It first partitions the requested weights into small packets and issues one packet transfer at a time to the runtime. This strategy prevents contention with other CPU-GPU transfers used by the compute pipeline, such as those triggered by PyTorch operations or attention-related data synchronization. Issuing all weight transfers at once would lead to head-of-line blocking, delaying latency-sensitive compute transfers. Empirically, a 100MB packet size strikes a good balance between transfer throughput and minimizing interference. The data mover is implemented in C++ as a PyTorch extension.

6.6 CPU Decode Attention

In §5.3, we highlight the importance of efficient vector unit utilization for CPU-side decode attention to fully exploit the CPU-GPU hybrid system. To this end, MoE-Lens implements a high-performance decode attention kernel using hand-written SIMD intrinsics. Figure 10 compares this implementation with an auto-vectorized baseline in terms of KV cache tokens attended per second. Although both leverage the AVX512 ISA on our test machine, they show stark performance differences. The auto-vectorized version under-utilizes vector units, falling short of system throughput requirements. In contrast, MoE-Lens's hand-optimized kernel—featuring manual vectorization, loop unrolling, and data prefetching—achieves 4.7× higher throughput in single-thread mode and 3.1× with full thread utilization, exceeding the system target. However, we observe that throughput gain saturates beyond 20 threads, likely due to memory controller contention.

7 Evaluation Methodology

Hardware Configuration and Test Environment. All experiments are conducted on a dual-socket server equipped with two Intel Platinum 8380 CPUs. Each socket has eight DDR4-3200 memory channels, totaling 750GB capacity, and is connected to two NVIDIA A40 GPUs. The measured aggregate CPU memory bandwidth per socket is approximately 150GB/s. To ensure consistency and isolate the experimental environment, we use `numactl` to restrict execution to a single CPU socket and a single GPU. The NVIDIA A40 GPU provides 48GB of memory; to simulate GPUs

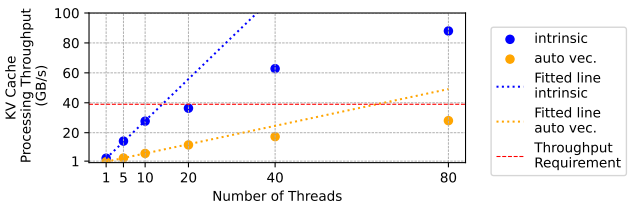


Figure 10: Performance comparison of decode flash attention implemented with intrinsic and auto vectorization. Trend lines are fitted using the first three data points to understand linear scaling. The throughput requirement is estimated assuming KV cache size is twice larger than the model size.

Dataset	Prefill Length		Max Generation Length	Category
	Avg.	Max		
MTBench [4]	98	450	32, 64, 128, 256	Multi-turn conversation
RAG [6]	926	1843	128	Retrieval-Augmented Q&A
AIME 2024 [22]	128	410	512	Math Problem Solving

Table 3: Model evaluation benchmarks used in evaluation.

with more constrained memory capacities, such as the NVIDIA T4 or L4 (16–24GB), we allocate 24–32GB of GPU memory with random tensors, thereby reducing the effective available memory for LLM serving. We configure the KV cache size in MoE-Lens to emulate systems with varying CPU memory capacities. Specifically, we experiment with KV cache sizes ranging from 70GB to 210GB, corresponding to machines with limited CPU-side memory.

Language Models. Following prior works [9, 19], we evaluate MoE-Lens using three diverse MoE models: `Mixtral8x7B` [1], `Mixtral8x22B` [2], and `DBRX` [14]. These models have 47B, 141B, and 132B parameters in BF16, with model sizes of 94GB, 282GB, and 264GB, respectively. We constrain the available GPU memory on the A40 to 16GB for `Mixtral8x7B`, and 24GB for both `Mixtral8x22B` and `DBRX`. Running `Mixtral8x7B` with a 70GB KV cache simulates a system with approximately 164GB of memory, while a 210GB KV cache corresponds to around 300GB. Similarly, using a 70GB KV cache for `Mixtral8x22B` and `DBRX` simulates systems with roughly 350GB of memory, and a 210GB KV cache corresponds to about 500GB.

LLM Benchmark/Datasets. We evaluate MoE-Lens using a diverse set of commonly used LLM model evaluation benchmarks. Following prior work, we use a replicated version of `MTBench` [4], which includes 80 high-quality multi-turn questions across various domains, to construct large batches for in-depth analysis. We also vary the maximum generation length across 32, 64, 128, and 256 tokens, consistent with previous studies. In addition, we assess MoE-Lens's performance on a RAG dataset [6] and the AIME dataset [22] to study its behavior under long-prompt and long-generation scenarios. The prefill length (p) and generation length (g) settings are summarized in Table 3. To ensure evaluations complete within reasonable time, we set the request batch size to 25k for $g = 32$, and 20k otherwise, when running `MTBench` with a 70GB KV cache. In all other settings, the request batch size is set to $5gq$, as defined in the performance model in §5.4.

Baselines. We compare MoE-Lens with two state-of-the-art baselines: `MoE-Lightning` [9] and `vllm` [26]. We use open-source implementations [8, 44] of both baselines for comparison.

- `MoE-Lightning` is the state-of-the-art throughput-oriented MoE inference system for resource-constrained environments. We evaluate `MoE-Lightning` with two CPU memory settings. The normal setting set the CPU memory size profile of `MoE-Lightning` as the sum of model size and KV cache size, plus an additional 30GB for execution overhead. We further evaluate a large CPU memory setting where we set the CPU memory capacity to 1.25× of the sum of the model and KV cache size.
- `vllm` is a widely used LLM serving system based on the idea of paged attention. We use its CPU offload option to run models larger than GPU memory size.

8 Evaluation Results

8.1 Overall Performance

System Generation Throughput. Figure 11 compares the generation throughput (*i.e.*, tokens generated per unit time) of vLLM, MoE-Lightening, and MoE-Lens. The vLLM baseline performs all computations, including GEMMs and attention, on the GPU while paging KV cache to and from CPU memory. Since model weights and KV cache exceed GPU memory capacity, vLLM is bottlenecked by the limited CPU–GPU PCIe bandwidth. MoE-Lightening mitigates this by offloading attention to the CPU, reducing I/O traffic and improving throughput. However, its modeling strategy overlooks key optimization opportunities, leaving substantial headroom before fully utilizing the hardware.

Figure 11 shows that MoE-Lens achieves an average throughput improvement of 4.6× (up to 12.4×) over MoE-Lightening. This gain stems from a key modeling insight overlooked in prior work: the role of CPU memory capacity in determining optimal workload scheduling. By incorporating this constraint, MoE-Lens identifies two critical decisions: (1) maximize the number of concurrent tokens on the GPU by fully utilizing CPU memory for KV cache usage, and (2) a prefill-decode overlap strategy that further expands effective KV cache capacity. In contrast, MoE-Lightening underutilizes CPU memory, as shown in Table 1, leading to fewer concurrent tokens and lower throughput. This disparity becomes more pronounced with a 210GB KV cache, where MoE-Lens delivers a higher average speedup of 6.4×, compared to 3.2× with a 70GB cache. The speedup is further improved to on average 5.3× when allowing MoE-Lens to terminate generation when the EOS token is reached.

With a large KV cache size (210GB), we observe that generation throughput increases with generation length up to a point. For example, running `MTBench` on `Mixtral-8x7B`, throughput improves as generation length increases from 32 to 128 tokens. However, at 256 generation tokens, KV cache becomes a bottleneck, causing throughput to drop. This rise-then-drop pattern is consistent across all models at 210GB KV cache. In contrast, with a 70GB KV cache, the rising trend disappears entirely, as even short generations saturate the limited cache, limiting throughput from the start.

System throughput decreases with longer generation lengths for a fixed prompt length. As analyzed in §5.2, longer generations reduce the PME, increasing CPU memory demand and reducing throughput under limited KV cache. When KV cache is sufficient, however, MoE-Lens approaches around 90% of GPU's computational limit: illustrated by the blue line with triangle markers in Figure 11 for $g_{\max} = 32/64$ with 210GB KV cache.

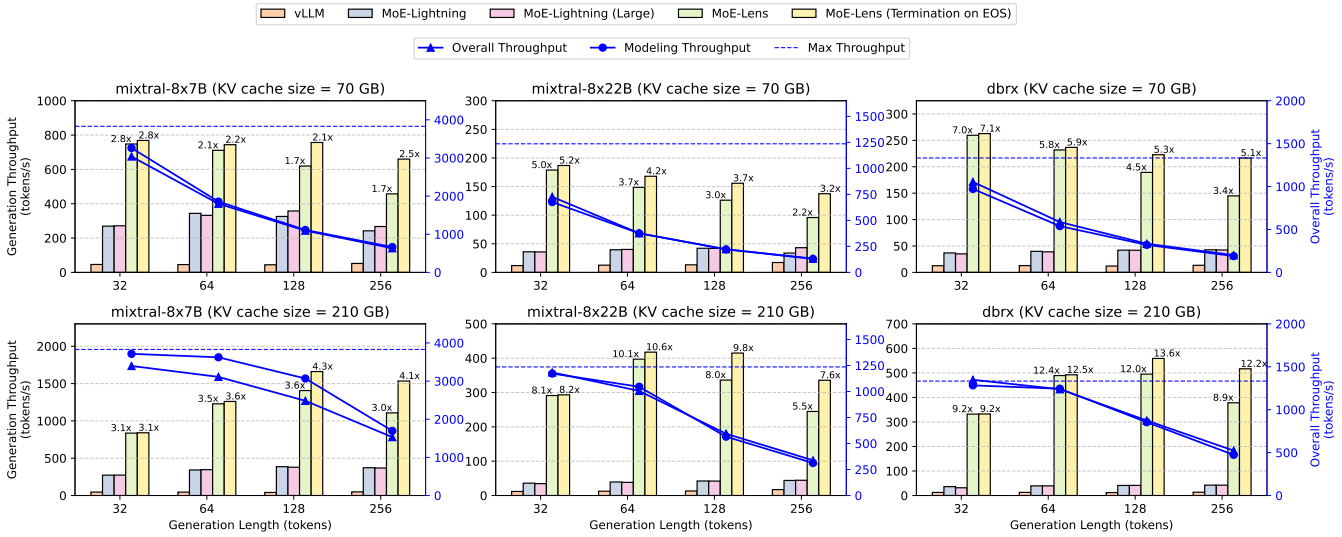


Figure 11: Overall performance of MoE-Lens compared to baselines. The text annotations represents the speedup of MoE-Lens against the best performance of MoE-Lightning. The secondary Y-axis shows the predicted and measured throughput of MoE-Lens, validating our theoretical model.

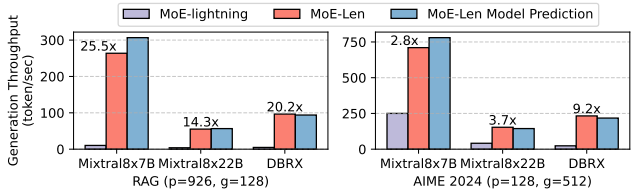


Figure 12: Performance comparison of MoE-Lens and MoE-Lightning for `RAG` and `AIME2024` datasets.

Figure 12 compares MoE-Lens's performance on the prefill-heavy `RAG` and generation-heavy `AIME2024` datasets. MoE-Lens achieves up to 25.5× (19.4× avg) speedup over MoE-Lightning on `RAG`, and up to 9.9× (4.7× avg) on `AIME2024`, demonstrating consistently high throughput across workloads of diverse characteristics.

MoE-Lens's Performance Model's Accuracy. On average, MoE-Lens's performance model (§5.4) predicts throughput with 94% accuracy, as shown in Figures 11 and 12. We estimate CPU-GPU IO bandwidth B_{IO} at 19.5GB/s, based on 1GB tensor transfers. Notably, MoE-Lens falls short of the model's prediction when running `Mixtral8x7B` with a 210GB KV cache, due to contention between CPU attention computation and IO, which delays weight transfers. We analyze this conflict further in §8.2.

8.2 Detailed Execution Status Analysis

The Execution Dynamic of MoE-Lens and the KV Cache Capacity Bottleneck. Figure 13 illustrates the detailed execution status of MoE-Lens when running `MTBench` on `Mixtral8x7B`. In the more resource-constrained 70GB KV cache setting, cache capacity significantly affects both overall system throughput and execution dynamics. When the maximum generation length is 32, the 70GB KV cache is sufficient to avoid sequence thrashing. Since no sequences are preempted, MoE-Lens achieves steady and high throughput in both prefill and decode phases. In this case, GPU utilization approaches around 90%, and system throughput is effectively distributed between prefill and decode phases, proportional to the ratio of prefill to decode tokens. As the maximum generation length increases, each sequence consumes more memory. Insufficient KV cache capacity causes sequence preemption, leading to observable fluctuations in the decode and prefill throughput curves.

When the maximum generation length is 64, the GPU is unable to prefill new sequences roughly half of the time due to insufficient KV cache capacity. During these preemption events, the prefill throughput drops to zero. As ongoing sequences complete their generation and release their KV cache blocks, the decode throughput also begins to decline. This decline continues until the rate of KV cache release exceeds the rate of token generation. Once enough KV cache blocks are freed, empty slots become available, allowing MoE-Lens's prefill scheduler to admit new sequences into the cache. This, in turn, increases the number of sequences being decoded in parallel, causing both prefill and decode throughput to rise again.

This phenomenon becomes more pronounced when the maximum generation length increases to 256. In this case, MoE-Lens performs prefilling for only a small fraction of the time, and the overall GPU utilization drops significantly. This is due to a limited number of active sequences in the decode stage, resulting in low PME. A similar trend is reflected in the per-forward-pass breakdown of GPU computation, CPU computation, and weight transfer I/O time. When the KV cache capacity is sufficient (*e.g.,* with a maximum generation length of 32), MoE-Lens effectively balances GPU computation and I/O time. However, when the generation length is 256, the KV cache becomes a bottleneck, leading to substantial GPU idle time.

The Effect of a Large KV Cache. By comparing the cases with 70GB and 210GB KV cache, we observe the impact of larger KV cache capacity on overall throughput and execution dynamics. When the generation length is 32, the 70GB cache is sufficient,

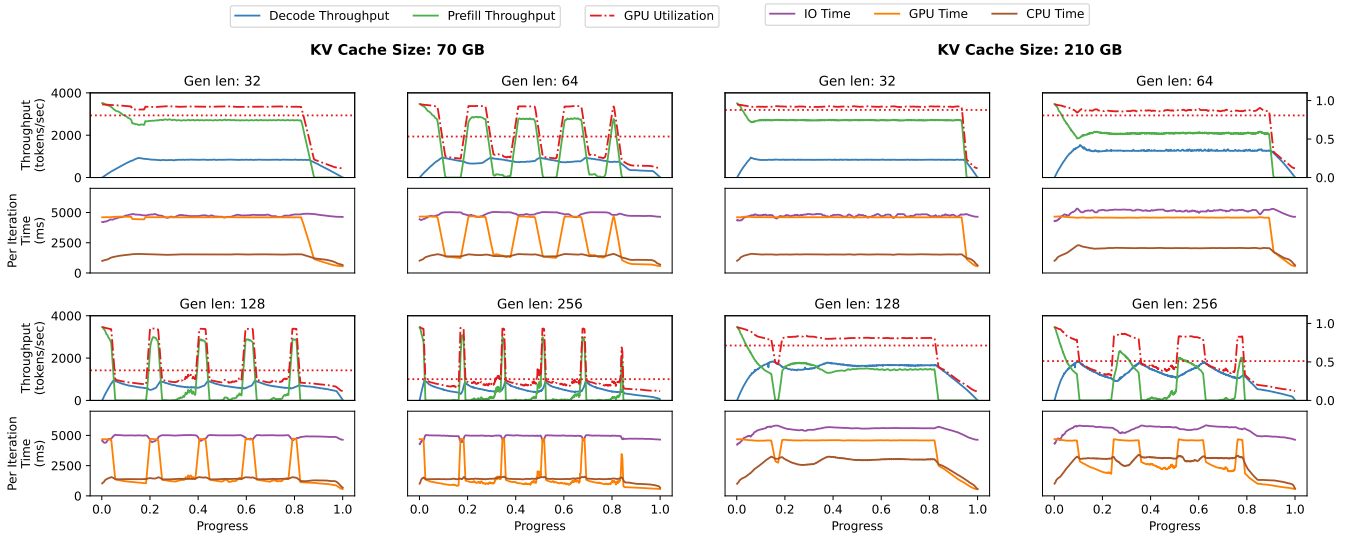


Figure 13: Execution status of MoE-Lens for running `MTBench` on `Mixtral8x7B`, with different maximum generation length and KV cache size. The first and third row illustrate how the throughput and GPU utilization changes with workload progress. The second and the fourth rows show the corresponding IO, GPU computation, and CPU attention time for each inference pass.

and increasing the cache size yields no further performance benefit. However, the advantage of a larger KV cache becomes evident at generation lengths of 64 or more. With 210GB of KV cache and a generation length of 64, decode and prefill throughput curves are noticeably smoother, reflecting fewer sequence preemptions due to ample cache capacity. This leads to higher overall throughput and better GPU utilization. A similar trend is observed for generation lengths of 128 and 256. Preemptions, if any, occur only in the early stages of execution, and the throughput stabilizes quickly. In the 256-token case, the duration of prefill stalls significantly decreases, further enhancing throughput. In both cases, the larger KV cache leads to substantial performance improvements.

The Bandwidth Competition between CPU Attention and CPU-GPU Weight Transfer. When the KV cache is large, the CPU attention computation must scan through a substantial number of KV cache blocks, concurrent with CPU-GPU weight transfers for GPU computation. These two operations contend for shared CPU-side resources, particularly memory bandwidth. When the maximum generation length is 256 and the KV cache size is 210GB, CPU attention time becomes significant relative to IO transfer time. This contention at the CPU memory controller slows down weight transfers, increasing the time to transfer weights from CPU to GPU from approximately 5 seconds to 6 seconds. Interestingly, IO transfer time decreases when preemption begins—fewer sequences remain in the decode stage, reducing the demand on CPU attention and thus easing the pressure on the memory subsystem.

9 Related Work

Resource-constrained LLM Inference. To enable LLM inference on resource-constrained hardware like PCs and low-end servers where the GPU memory capacity is limited, prior works focus on offloading GPU states to CPU [18, 19, 38, 39, 45, 46] and disks [3, 34, 38]. MoE-lightning [9] is the state-of-the-art MoE inference system for resource-constrained environments. It provides a CPU-GPU IO-aware performance model, HRM, on offloading KV cache and attention computation to CPUs. On comparison, MoE-Lens provides a more comprehensive and fine-grained performance model that goes beyond CPU–GPU I/O considerations by accounting for CPU architecture resources and sequence-level scheduling dynamics, and a system significantly outperforms MoE-lightning.

Online LLM Inference Optimization. Online LLM systems with strict latency constraints are usually distributed [33, 36, 41, 42, 47]. Prior works focus on diverse optimization objectives, including but not limited to latency/throughput [23, 25, 26, 35, 51, 52], cost [21, 43], energy efficiency [32, 40, 41]. The online LLM serving system usually employs prefill decode disaggregation [33, 36] to achieve low latency for token generation. Techniques like continuous batching [12] also let the new sequences prefill while others are decoding, but they do not discuss their dynamics for inference under resource-constrained environments, and focus on online serving. In contrast, MoE-Lens provides a detailed performance model to quantify the impact of prefill-decode overlap in resource-constrained environments, and demonstrates how to achieve this with weight transfer and attention computation on the CPU in a balanced manner. This differs fundamentally from continuous batching, whose techniques cannot be directly applied in our setting due to the differing coordination and resource constraints.

Algorithmic Techniques for LLM Acceleration. Another line of research in accelerating LLM inference focuses on trading off the generation accuracy with the LLM inference speed, such as quantization [17, 28, 50], kv cache compression [7, 49], developing attention variants [29, 48], etc. MoE-Lens focuses on improving the hardware resource utilization for MoE inference in resource-constrained environments, without algorithmic changes.

10 Conclusion

This paper uncovered a significant opportunity to improve performance in resource-constrained, batch-processing of MoE LLM

inference. In CPU–GPU hybrid execution, we showed that *effectively utilizing CPU memory capacity* is key to boosting end-to-end throughput, as it enables more tokens to be processed on the GPU. Building on this insight, we developed a theoretical performance model that captures architectural heterogeneity across CPU, GPU, and their interconnects, as well as workload heterogeneity between prefill and decode stages. The model not only identifies critical performance bottlenecks but also predicts overall performance with an average validation accuracy of 94%. Using these insights, we designed an inference system that achieves a 4.6× average speedup (up to 25.5×) over the state-of-the-art.

Acknowledgments

This work was supported in part by Advanced Micro Devices, Inc. under the "Funding Academic Research" and the AMD AI & HPC Cluster Program.

References

[1] Mistral AI. 2023. Mixtral-8x7B-Instruct-v0.1. https://huggingface.co/mistralai/Mixtral-8x7B-Instruct-v0.1. Accessed: 2025-04-10.

[2] Mistral AI. 2025. Mixtral-8x22B-Instruct-v0.1. https://huggingface.co/mistralai/Mixtral-8x22B-Instruct-v0.1. Accessed: 2025-04-10.

[3] Keivan Alizadeh, Iman Mirzadeh, Dmitry Belenko, Karen Khatamifard, Minsik Cho, Carlo C Del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. LLM in a flash: Efficient Large Language Model Inference with Limited Memory. arXiv:2312.11514 [cs.CL] https://arxiv.org/abs/2312.11514

[4] Ge Bai, Jie Liu, Xingyuan Bu, Yancheng He, Jiaheng Liu, Zhanhui Zhou, Zhuoran Lin, Wenbo Su, Tiezheng Ge, Bo Zheng, and Wanli Ouyang. 2024. MT-Bench-101: A Fine-Grained Benchmark for Evaluating Large Language Models in Multi-Turn Dialogues. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 7421–7454. https://doi.org/10.18653/v1/2024.acl-long.401

[5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen Technical Report. arXiv:2309.16609 [cs.CL] https://arxiv.org/abs/2309.16609

[6] Neural Bridge. 2024. RAG Dataset 12000. https://huggingface.co/datasets/neural-bridge/rag-dataset-12000. Accessed: 2025-04-10.

[7] Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, and Wen Xiao. 2024. PyramidKV: Dynamic KV Cache Compression based on Pyramidal Information Funneling. arXiv:2406.02069 [cs.CL] https://arxiv.org/abs/2406.02069

[8] Shiyi Cao. 2025. Artifact Evaluation Repository for ASPLOS 2025. https://github.com/caoshiyi/artifacts/tree/asplos25. Accessed: 2025-04-11.

[9] Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, Joseph E Gonzalez, Matei Zaharia, and Ion Stoica. 2024. Moe-lightning: High-throughput moe inference on memory-constrained gpus. *arXiv preprint arXiv:2411.11217* (2024).

[10] Xinyun Chen, Petros Maniatis, Rishabh Singh, Charles Sutton, Hanjun Dai, Max Lin, and Denny Zhou. 2021. SpreadsheetCoder: Formula Prediction from Semi-structured Context. arXiv:2106.15339 [cs.SE] https://arxiv.org/abs/2106.15339

[11] Damai Dai, Chengqi Deng, Chenggang Zhao, R. X. Xu, Huazuo Gao, Deli Chen, Jiashi Li, Wangding Zeng, Xingkai Yu, Y. Wu, Zhenda Xie, Y. K. Li, Panpan Huang, Fuli Luo, Chong Ruan, Zhifang Sui, and Wenfeng Liang. 2024. DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models. arXiv:2401.06066 [cs.CL] https://arxiv.org/abs/2401.06066

[12] Cade Daniel, Chen Shen, Eric Liang, and Richard Liaw. 2023. How Continuous Batching Enables 23x Throughput in LLM Inference While Reducing p50 Latency. https://www.anyscale.com/blog/continuous-batching-llm-inference. Accessed: 2025-04-11.

[13] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. arXiv:2205.14135 [cs.LG] https://arxiv.org/abs/2205.14135

[14] Databricks. 2024. DBRX Instruct. https://huggingface.co/databricks/dbrx-instruct. Accessed: 2025-04-10.

[15] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[16] DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Haowei Zhang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Li, Hui Qu, J. L. Cai, Jian Liang, Jianzhong Guo, Jiaqi Ni, Jiashi Li, Jiawei Wang, Jin Chen, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Junxiao Song, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Xu, Leyi Xia, Liang Zhao, Litong Wang, Liyue Zhang, Meng Li, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Ning Tian, Panpan Huang, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qinyu Chen, Qiushi Du, R. J. Chen, R. L. Jin, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runxin Xu, Ruoyu Zhang, Ruyi Chen, S. S. Li, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaoqing Wu, Shengfeng Ye, Shengfeng Ye, Shirong Ma, Shiyu Wang, Shuang Zhou, Shuiping Yu, Shunfeng Zhou, Shuting Pan, T. Wang, Tao Yun, Tian Pei, Tianyu Sun, W. L. Xiao, Wangding Zeng, Wanjia Zhao, Wei An, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, X. Q. Li, Xiangyue Jin, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojin Shen, Xiaokang Chen, Xiaokang Zhang, Xiaosha Chen, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xinnan Song, Xinxia Shan, Xinyi Zhou, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. X. Zhu, Yang Zhang, Yanhong Xu, Yanhong Xu, Yanping Huang, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Li, Yaohui Wang, Yi Yu, Yi Zheng, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Ying Tang, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yu Wu, Yuan Ou, Yuchen Zhu, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yukun Zha, Yunfan Xiong, Yunxian Ma, Yuting Yan, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z. F. Wu, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhen Huang, Zhen Zhang, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhipeng Xu, Zhiyu Wu, Zhongyu Zhang, Zhuoshu Li, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Ziyi Gao, and Zizheng Pan. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[17] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2023. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. arXiv:2306.03078 [cs.CL] https://arxiv.org/abs/2306.03078

[18] 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

[19] Zhiyuan Fang, Yuegui Huang, Zicong Hong, Yufeng Lyu, Wuhui Chen, Yue Yu, Fan Yu, and Zibin Zheng. 2025. Klotski: Efficient Mixture-of-Expert Inference

via Expert-Aware Multi-Batch Pipeline. arXiv:2502.06888 [cs.LG] https://arxiv.org/abs/2502.06888

[20] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[21] Tao Huang, Pengfei Chen, Kyoka Gong, Jocky Hawk, Zachary Bright, Wenxin Xie, Kecheng Huang, and Zhi Ji. 2024. ENOVA: Autoscaling towards Cost-effective and Stable Serverless LLM Serving. arXiv:2407.09486 [cs.DC] https://arxiv.org/abs/2407.09486

[22] Maxwell Jia. 2024. AIME 2024 Dataset. https://huggingface.co/datasets/Maxwell-Jia/AIME_2024. Accessed: 2025-04-10.

[23] Jordan Juravsky, Bradley Brown, Ryan Ehrlich, Daniel Y. Fu, Christopher Ré, and Azalia Mirhoseini. 2024. Hydragen: High-Throughput LLM Inference with Shared Prefixes. arXiv:2402.05099 [cs.LG] https://arxiv.org/abs/2402.05099

[24] Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci. 2025. Fiddler: CPU-GPU Orchestration for Fast Inference of Mixture-of-Experts Models. arXiv:2402.07033 [cs.LG] https://arxiv.org/abs/2402.07033

[25] Junsoo Kim, Hunjong Lee, Geonwoo Ko, Gyubin Choi, Seri Ham, Seongmin Hong, and Joo-Young Kim. 2025. ADOR: A Design Exploration Framework for LLM Serving with Enhanced Latency and Throughput. arXiv:2503.04253 [cs.AR] https://arxiv.org/abs/2503.04253

[26] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180 [cs.LG] https://arxiv.org/abs/2309.06180

[27] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic Evaluation of Language Models. arXiv:2211.09110 [cs.CL] https://arxiv.org/abs/2211.09110

[28] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 87–100. https://proceedings.mlsys.org/paper_files/paper/2024/file/42a452cbafa9dd64e9ba4aa95cc1ef21-Paper-Conference.pdf

[29] Di Liu, Meng Chen, Baotong Lu, Huiqiang Jiang, Zhenhua Han, Qianxi Zhang, Qi Chen, Chengruidong Zhang, Bailu Ding, Kai Zhang, Chen Chen, Fan Yang, Yuqing Yang, and Lili Qiu. 2024. RetrievalAttention: Accelerating Long-Context LLM Inference via Vector Retrieval. arXiv:2409.10516 [cs.LG] https://arxiv.org/

abs/2409.10516
[30] Shu Liu, Asim Biswal, Amog Kamsetty, Audrey Cheng, Luis Gaspar Schroeder, Liana Patel, Shiyi Cao, Xiangxi Mo, Ion Stoica, Joseph E. Gonzalez, and Matei Zaharia. 2025. Optimizing LLM Queries in Relational Data Analytics Workloads. arXiv:2403.05821 [cs.LG] https://arxiv.org/abs/2403.05821
[31] Avanika Narayan, Ines Chami, Laurel Orr, Simran Arora, and Christopher Ré. 2022. Can Foundation Models Wrangle Your Data? arXiv:2205.09911 [cs.LG] https://arxiv.org/abs/2205.09911
[32] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, and Ricardo Bianchini. 2024. Characterizing Power Management Opportunities for LLMs in the Cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3* (La Jolla, CA, USA) *(ASPLOS '24)*. Association for Computing Machinery, New York, NY, USA, 207–222. https://doi.org/10.1145/3620666.3651329
[33] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. 118–132. https://doi.org/10.1109/ISCA59077.2024.00019
[34] Jie Peng, Zhang Cao, Huaizhi Qu, Zhengyu Zhang, Chang Guo, Yanyong Zhang, Zhichao Cao, and Tianlong Chen. 2024. Harnessing Your DRAM and SSD for Sustainable and Accessible LLM Inference with Mixed-Precision and Multi-level Caching. arXiv:2410.14740 [cs.LG] https://arxiv.org/abs/2410.14740
[35] Yifan Qiao, Shu Anzai, Shan Yu, Haoran Ma, Yang Wang, Miryung Kim, and Harry Xu. 2024. ConServe: Harvesting GPUs for Low-Latency and High-Throughput Large Language Model Serving. arXiv:2410.01228 [cs.DC] https://arxiv.org/abs/2410.01228
[36] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. arXiv:2407.00079 [cs.DC] https://arxiv.org/abs/2407.00079
[37] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. arXiv:1701.06538 [cs.LG] https://arxiv.org/abs/1701.06538
[38] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Daniel Y. Fu, Zhiqiang Xie, Beidi Chen, Clark Barrett, Joseph E. Gonzalez, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. arXiv:2303.06865 [cs.LG] https://arxiv.org/abs/2303.06865
[39] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. arXiv:2312.12456 [cs.LG] https://arxiv.org/abs/2312.12456
[40] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Esha Choukse, Haoran Qiu, Rodrigo Fonseca, Josep Torrellas, and Ricardo Bianchini. 2025. TAPAS: Thermal- and Power-Aware Scheduling for LLM Inference in Cloud Platforms. arXiv:2501.02600 [cs.DC] https://arxiv.org/abs/2501.02600
[41] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2024. DynamoLLM: Designing LLM Inference Clusters for Performance and Energy Efficiency. arXiv:2408.00741 [cs.AI] https://arxiv.org/abs/2408.00741
[42] Biao Sun, Ziming Huang, Hanyu Zhao, Wencong Xiao, Xinyi Zhang, Yong Li, and Wei Lin. 2024. Llumnix: Dynamic Scheduling for Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 173–191. https://www.usenix.org/conference/osdi24/presentation/sun-biao
[43] The AIBrix Team, Jiaxin Shan, Varun Gupta, Le Xu, Haiyang Shi, Jingyuan Zhang, Ning Wang, Linhui Xu, Rong Kang, Tongping Liu, Yifei Zhang, Yiqing Zhu, Shuowei Jin, Gangmuk Lim, Binbin Chen, Zuzhi Chen, Xiao Liu, Xin Chen, Kante Yin, Chak-Pong Chung, Chenyu Jiang, Yicheng Lu, Jianjun Chen, Caixue Lin, Wu Xiang, Rui Shi, and Liguang Xie. 2025. AIBrix: Towards Scalable, Cost-Effective Large Language Model Inference Infrastructure. arXiv:2504.03648 [cs.DC] https://arxiv.org/abs/2504.03648
[44] vLLM Team. 2023. vLLM: A High-Throughput and Memory-Efficient Inference Engine for LLMs. https://github.com/vllm-project/vllm. Accessed: 2025-04-11.
[45] Tairan Xu, Leyang Xue, Zhan Lu, Adrian Jackson, and Luo Mai. 2025. MoE-Gen: High-Throughput MoE Inference on a Single GPU with Module-Based Batching. arXiv:2503.09716 [cs.DC] https://arxiv.org/abs/2503.09716
[46] 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
[47] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. USENIX Association, Carlsbad, CA, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu
[48] Jingyang Yuan, Huazuo Gao, Damai Dai, Junyu Luo, Liang Zhao, Zhengyan Zhang, Zhenda Xie, Y. X. Wei, Lean Wang, Zhiping Xiao, Yuqing Wang, Chong Ruan, Ming Zhang, Wenfeng Liang, and Wangding Zeng. 2025. Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. arXiv:2502.11089 [cs.CL] https://arxiv.org/abs/2502.11089
[49] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. 2023. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. arXiv:2306.14048 [cs.LG] https://arxiv.org/abs/2306.14048
[50] Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. 2024. Atom: Low-Bit Quantization for Efficient and Accurate LLM Serving. In *Proceedings of Machine Learning and Systems*, P. Gibbons, G. Pekhimenko, and C. De Sa (Eds.), Vol. 6. 196–209. https://proceedings.mlsys.org/paper_files/paper/2024/file/5edb57c05c81d04beb716ef1d542fe9e-Paper-Conference.pdf
[51] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. 2024. SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104 [cs.AI] https://arxiv.org/abs/2312.07104
[52] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. USENIX Association, Santa Clara, CA, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin