MESO: Memory-Efficient Speculative Offloading for Large Language Model Inference

Qingxiao Zhang, Xiaopeng Li*, Bin Ji*, Xiaodong Liu*, Jie Yu*, Long Peng, Hao Xu
College of Computer Science and Technology, National University of Defense Technology, Changsha, China
{zhangqxnudt, xiaopengli, jibin, liuxiaodong, yj, penglong, xuhao19}@nudt.edu.cn

***Abstract*—In resource-constrained devices, Large Language Model (LLM) inference faces significant challenges due to limited GPU memory resources. Existing studies, such as SpecOffload, combine offloading with speculative decoding to utilize a draft model for generating candidate tokens, thereby improving LLM generation throughput. However, they require the draft model to reside permanently in GPU memory. In resource-constrained devices, it tends to introduce high fixed overheads, resulting in low memory efficiency, which can be measured by the throughput gain per unit of GPU memory usage. To address this limitation, we propose MESO, a framework that specializes in improving memory efficiency. Specifically, MESO achieves high memory efficiency by offloading the draft model, enabling fine-grained asynchronous component loading, and leveraging pinned memory optimization. Experimental results show that compared to the selected baselines, MESO achieves 1.35× memory efficiency on average across various scenarios, and delivers up to 2.25× memory efficiency. Additionally, MESO reduces peak GPU memory usage by more than 50% compared to SpecOffload. Quantitative and qualitative analyses further validate the effectiveness of MESO.**

***Index Terms*—large language model inference, mixture-of-experts, resource-constrained device, CPU offloading, speculative decoding**

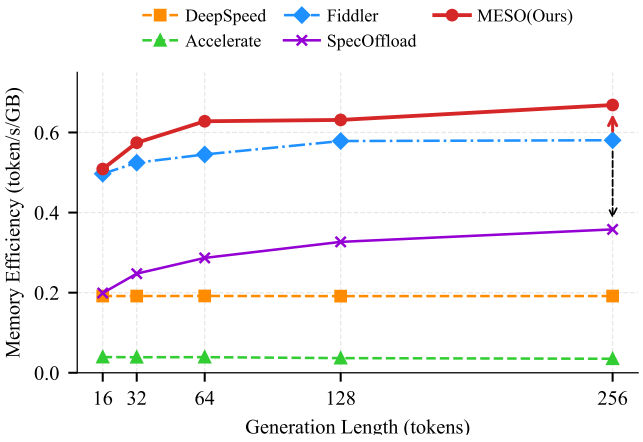

Fig. 1. Memory efficiency comparison.

I. Introduction

The booming development of Large Language Models (LLMs) has driven their adoption across various intelligent applications. Despite the availability of cloud-based API services, there is a growing imperative to deploy LLMs locally on resource-constrained devices. This shift is driven by critical needs for data privacy, latency-sensitive interaction, and operational cost reduction [1]–[3]. However, the democratization of LLMs is strictly gated by the memory wall, which is the widening gap between the rapid growth of model parameters and the stagnant capacity of GPU memory.

In this work, we explicitly define "resource-constrained devices" as scenarios characterized by a hard bottleneck in GPU memory capacity, rather than limitations in host compute power or RAM. This scope targets a widespread class of hardware, such as high-end workstations equipped with powerful CPUs and sufficient system memory, yet whose GPU memory is insufficient to hold the target model. In such environments, leveraging host resources for heterogeneous orchestration becomes the only viable path. Deploying LLMs on such devices faces severe constraints that limit deployment feasibility and real-time response capabilities [1]. This is particularly true for Mixture-of-Experts (MoE) architectures [4]. While MoE models achieve efficient scaling by dynamically activating a subset of experts, they incur higher memory footprints and complex scheduling requirements [5], [6]. For instance, deploying the Mixtral-8x7B [7] model requires at least 87GB of GPU memory for full GPU execution, whereas typical GPUs (*e.g.*, the A100 40GB version) have limited capacity, making direct deployment nearly impossible. Furthermore, the dynamic sparsity of MoE disrupts the spatial locality of memory access, causing severe I/O thrashing when naive offloading strategies are applied. Therefore, achieving efficient MoE inference with low memory usage on these devices is of significant research value.

To address these challenges, researchers have proposed various optimization methods. For example, Fiddler [8] intelligently utilizes CPU and GPU resources to dynamically select optimal execution strategies, effectively enabling large MoE models to run on resource-constrained devices while improving inference speed without compromising accuracy. SpecOffload [9] further enhances throughput by embedding cloud-oriented Speculative Decoding (SD) [10] into the offloading mechanism. However, we demonstrate a critical common limitation existing in these optimization schemes. Whether it is the academic framework SpecOffload, or the industrial standard system vLLM [11], they predominantly adopt an Accelerator-Resident strategy. The former forces the draft model to reside in GPU memory, while the latter

*Corresponding author.

similarly requires full residency of the draft model when SD is enabled. While this "trading space for time" strategy is effective in cloud environments with moderate resources, it locks valuable memory resources on constrained devices. Consequently, these methods perform suboptimal in terms of "Memory Efficiency", which is defined as the throughput gain per unit of GPU memory usage (as shown in Fig. 1), limiting their practical value in resource-constrained devices.

To resolve the above issue, we propose MESO, a **M**emory-**E**fficient **S**peculative **O**ffloading framework designed to maximize memory efficiency for cost-effective deployment. To achieve this, MESO first eliminates the persistent GPU memory occupancy of the draft model by offloading it entirely to host memory and employing an on-demand layer loading strategy, thereby drastically reducing peak GPU memory usage. Second, to mitigate the latency overhead of data transfer, we transform the loading of MoE components (gating networks and experts) from a synchronous blocking process into an asynchronous parallel one by launching dedicated I/O threads for each component. Finally, we utilize pinned memory for frequently accessed components to accelerate GPU data access and minimize latency.

We extensively evaluate MESO on the Mixtral-8x7B model across multiple datasets (including HumanEval, SummEval, and Samsum) and varying batch sizes. Experimental results demonstrate that MESO achieves superior memory efficiency, achieving 1.35× on average across various scenarios, and delivers up to 2.25×. Additionally, MESO reduces peak GPU memory usage by more than 50% compared to SpecOffload. Crucially, under identical GPU memory constraints, MESO delivers the best throughput performance. Finally, we verify the compatibility of MESO with INT4 quantized models, further confirming the framework's generality and extensibility.

The main contributions can be summarized as follows:

(1) **New Evaluation Metric:** We introduce "Memory Efficiency" as a complementary metric to quantify the cost-effectiveness of GPU memory utilization in resource-constrained devices, defined as throughput gain per unit of GPU memory usage.

(2) **New Framework Design:** We propose MESO, which eliminates the persistent GPU footprint of draft models. By combining full draft model offloading with asynchronous I/O scheduling and pinned memory mechanisms, it maximizes memory efficiency.

(3) **Extensive Evaluation:** Experiments show that MESO achieves significant memory efficiency (up to **2.25**× that of the best baseline) and enables high-performance inference of massive MoE models on hardware with strictly limited GPU memory.

II. Related work

A. Mixture-of-Experts

LLMs based on the MoE architecture have demonstrated superior performance in various applications. Unlike dense Transformer models, MoE models introduce sparsity in the feed-forward layers through expert subnetworks and gating mechanisms [12]. Each MoE layer contains multiple expert modules matching the shape of the feed-forward layer, and a gating network determines which experts to activate for each input. Although a single MoE layer typically contains dozens to hundreds of experts (*e.g.*, Mixtral-8x7B contains 8 experts), only a small subset (*e.g.*, the top-2 experts) is activated during inference. This dynamic sparse activation pattern brings unique challenges to memory management and computation scheduling, especially in resource-constrained devices.

B. Speculative Decoding

Speculative decoding has become an effective technique for accelerating LLM inference [10], [13]–[15]. It uses a draft model to pre-generate a sequence of candidate tokens, which are then verified in parallel by a large target model, thereby reducing the number of target model calls. Recent advancements have diversified the implementation of speculative decoding. Frameworks like Medusa [14] and Eagle [15] introduce additional decoding heads or lightweight feature-level extrapolation to eliminate the need for a separate draft model. While these methods reduce the computational overhead, they typically assume the entire model fits in GPU memory, making it primarily suitable for scenarios with ample hardware resources, thus posing significant challenges for resource-constrained settings. Adapting these advanced speculative techniques to an offloading-based context remains an underexplored frontier.

C. Heterogeneous Architecture Deployment

Efficiently deploying MoE models in resource-constrained devices remains highly challenging, primarily due to their large parameter scale. Existing methods attempt to address this by offloading computation to CPU resources, yet they still fall short in achieving efficient MoE inference.

Offloading is a mainstream strategy. It keeps less frequently used model parameters in CPU memory or storage and transfers them to the GPU on demand, helping to overcome memory bottlenecks [8], [16]–[19]. Nevertheless, such approaches often incur substantial latency overhead because expert weights must be frequently transferred between CPU and GPU via the PCIe, whose bandwidth is significantly lower than that of direct memory access. This results in a difficult trade-off between memory savings and inference speed. For example, Fiddler optimizes MoE inference through intelligent CPU-GPU resource orchestration and dynamic execution policies, yet its performance remains constrained by I/O bandwidth limitations, as well as CPU compute and memory access latency [8].

Recent work, such as SpecOffload, combines speculative decoding with model offloading. It offloads the target model to CPU memory while keeping the draft model permanently resident on the GPU, exploiting idle I/O waiting periods to perform speculative computations and thereby achieving high throughput. However, this design suffers from two critical issues: (1) The resident draft model incurs a high fixed GPU memory overhead; (2) Offloading the target model creates a

performance bottleneck during verification. These issues lead to suboptimal Memory Efficiency (throughput gain per unit of GPU memory usage), which is critical for resource-constrained devices.

Our proposed MESO is the first to explicitly define "Memory Efficiency" as a key optimization goal for these scenarios. By using dynamic draft model offloading, fine-grained asynchronous I/O, and pinned memory, we systematically optimize both GPU memory usage and inference speed.

D. Model Compression and Quantization

Model compression techniques, particularly quantization, serve as a pivotal strategy for deploying LLMs on resource-constrained devices. Methods like GPTQ [20] and AWQ [21] reduce the precision of model weights (*e.g.*, FP16 to INT4), shrinking the model size by up to 75% with negligible accuracy degradation. While these techniques significantly lower the static memory requirement, they do not alter the execution mechanism; a quantized model may still exceed the GPU memory capacity of the devices. Our work treats quantization as an orthogonal optimization dimension. By integrating MESO with quantized models, we demonstrate that dynamic offloading can work in tandem with static compression to achieve extreme memory efficiency, pushing the boundaries of what is deployable on consumer-grade hardware.

III. Methodology

A. Memory Efficiency

In data center scenarios, where resources are nearly unlimited, LLM optimization usually focuses on throughput or latency. However, on memory-constrained devices, performance depends on the trade-off between speed and resource consumption. Existing evaluations often report throughput and peak GPU memory usage separately, which fails to reflect the true usability of an LLM inference engine under a strict memory budget. Moreover, it does not answer a key question: Under a given hardware memory constraint, which approach offers the best cost-effectiveness of GPU memory utilization?

Inspired by the concept of memory efficiency metrics in high-performance computing and FPGA research, where Chen and Prasanna defined memory efficiency in FPGA sorting architectures as "the ratio of system throughput to the on-chip storage capacity use" [22], we adapt this concept to the LLM inference. We define Memory Efficiency (η) as:

$$\eta = \frac{T}{M_{\text{peak}}}, \tag{1}$$

where T represents throughput, i.e., the number of tokens generated by the model per unit of time; and M_{peak} represents the peak GPU memory usage. The essence of this metric is to treat GPU memory as a scarce cost, intuitively measuring the "return on investment" per unit of cost.

It is important to note that memory efficiency is not a substitute for throughput, but rather a conditional expression of throughput under resource constraints. High memory efficiency implies that the framework can achieve higher effective throughput given a specific memory limit. In resource-constrained devices, existing work generally lacks attention to this dimension. We not only explicitly propose this new metric, but establishe maximizing memory efficiency as the primary principle guiding the entire design of the MESO framework.

B. The MESO Framework

To maximize memory efficiency, we propose the MESO, which incorporates three key optimization techniques, as illustrated in Fig. 2: (1) A Draft Model Offloading mechanism to eliminate the fixed memory overhead; (2) An Asynchronous Prefetch Pipeline tailored for the MoE gating-expert structure; and (3) Load-to-GPU Pinning for high-frequency access components (such as MLPs). The following sections will detail the design of these mechanisms and their synergistic operation.

1) Draft Model Offloading: SpecOffload keeps the draft model resident in GPU memory to support low-latency speculative execution. However, this design introduces a fixed memory overhead that is incompressible for resource-constrained devices, significantly constraining its memory efficiency. To address this, we use a complete offloading strategy: all parameters of the draft model, including MLP weights and Layer Normalization parameters, as well as its runtime states (KV Cache), are fully offloaded to CPU memory. They are dynamically loaded into the GPU layer-by-layer only when needed during speculative decoding.

However, executing the full layer forward pass on the GPU mandates the migration of the entire KV cache (of size $O(b \cdot s \cdot d)$, where b is batch size, s is sequence length, and d is hidden dimension) from host to device at every step. This incurs not only GPU memory usage spikes but also prohibitive I/O latency, which negates the speedup gains of speculative decoding. To strictly limit these overheads, we introduce Attention Computation Delegation, which offloads the draft model's attention operations to the CPU. In this design, the GPU transfers only the current activations (Query, $O(b \cdot d)$) to the CPU. The CPU then computes the attention output using the host-resident KV cache and returns it to the GPU for the subsequent MLP block. This reduces data transfer volume to $1/s$ of the original [17], [23], [24], effectively relieving I/O pressure in long-context scenarios. Furthermore, to mask the loading latency of MLP weights, we initiate an asynchronous prefetch thread concurrent with the CPU attention computation. This pipeline preloads the MLP weights of the next layer from CPU memory to GPU memory, overlapping computation with communication and thereby minimizing the latency penalty introduced by offloading.

Note that while this strategy slightly increases latency per step, it massively reduces GPU memory usage. Since resource-constrained deployment is bounded by hard memory limits, this trade-off is necessary to make deployment feasible and enhance resource cost-effectiveness.

2) Fine-grained Component Loading: In standard offloading schemes like SpecOffload, target model execution relies

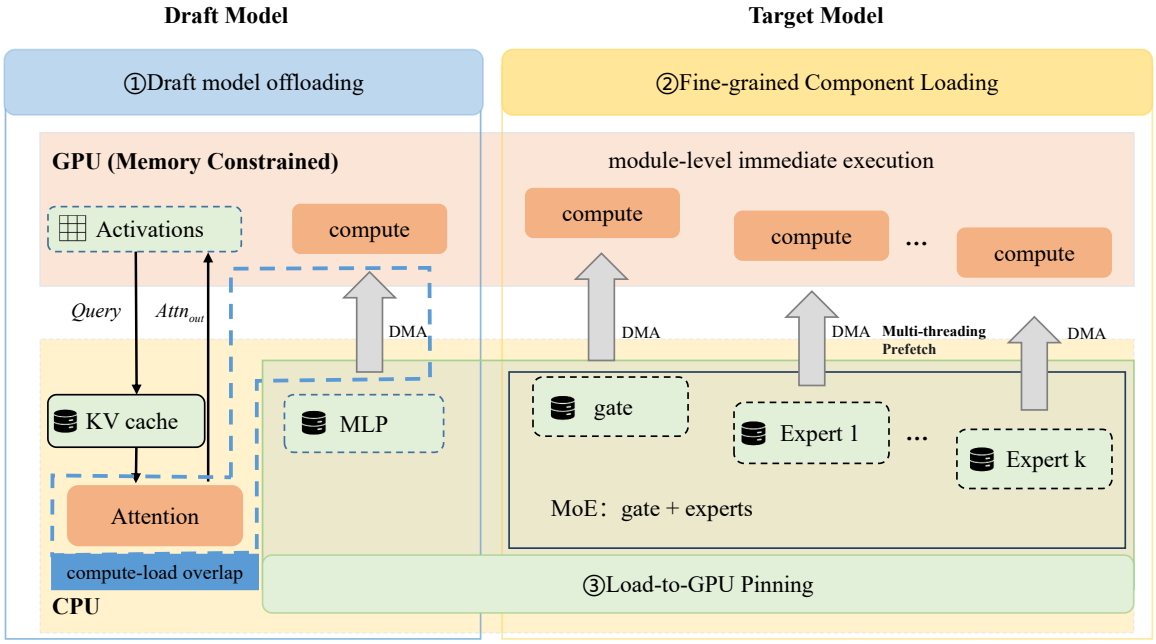

Fig. 2. The MESO execution flow. To maximize memory efficiency, the framework delegates draft model attention to the CPU, enabling computation-transfer overlap (blue dashed region) that hides I/O costs. On the target side, a fine-grained asynchronous pipeline fetches only top-k experts on demand. Underlying these operations, a Pinned Memory pool (green container) manages all weight tensors to ensure zero-copy DMA transmission.

on layer-level synchronous loading: the system blocks until the gating network and all expert weights are fully loaded before starting forward computation. However, this coarse-grained approach is inefficient for MoE architectures. Since only the top-k experts are active per token, loading all experts is wasteful. Moreover, the lightweight gating computation can be decoupled and executed early to identify the active experts.

To leverage this sparsity, MESO introduces Fine-grained Asynchronous Loading. At the start of each layer's computation, the system prioritizes loading and executing the gating network. Due to its negligible parameter size ($\approx$ 32KB) compared to the massive expert weights, the top-k active experts are identified almost instantly. The system then assigns dedicated I/O threads to prefetch the weights of these specific experts in parallel. Crucially, as soon as an individual expert's weights are ready, the GPU immediately performs computation without waiting for others. This transforms the traditional serial "load–then–compute" sequence into a highly overlapped computation–loading pipeline, effectively masking I/O latency and maximizing inference efficiency.

3) Load-to-GPU Pinning: In dynamic loading, transferring data from default paged memory requires an intermediate CPU copy to a driver-managed staging buffer before the DMA transfer to the GPU. This incurs extra host memory copy overhead and introduces significant latency jitter due to frequent TLB lookups, which destabilizes the asynchronous loading pipeline.

To eliminate this bottleneck, MESO enforces the use of Pinned Memory (page-locked memory) for all high-frequency components, including draft model MLP weights and target MoE parameters. Specifically, we pre-allocate physically contiguous pinned buffers in host memory and keep relevant weights resident in these regions. This enables zero-copy asynchronous transmission via `tensor.to(device, non_blocking = True)`, allowing the GPU's DMA engine to fetch data directly from pinned memory without intermediate copies or runtime page-locking. This optimization minimizes component transfer latency, providing the robust infrastructure required for effective asynchronous loading and ensuring consistent high memory efficiency.

4) Synergy with Quantization: The architecture of MESO is inherently orthogonal to model compression techniques. To be specific, whereas dynamic offloading optimizes runtime resource scheduling, INT4 quantization (*e.g.*, GPTQ) addresses static weight representation. This characteristic enables the system to seamlessly integrate pre-quantized models, compressing the weight volume to one-quarter that of FP16 with negligible loss in task accuracy, thereby significantly reducing both the GPU memory usage and the cross-device I/O overhead.

Leveraging this orthogonality, we construct a tunable memory-speed deployment spectrum. By independently configuring the quantization granularity (FP16 or INT4) for both the draft and target MoE model, the system can flexibly adapt to three representative modes: High-Precision, Balanced, and Minimal-Resource. This allows users to instantiate the optimal setting on demand, strictly tailored to their device's specific memory capacity and latency constraints. The synergistic optimization of quantization and offloading further boosts memory efficiency. We present more details in section IV-D.

IV. Experiment

A. Experimental Setup

Implementation. We implemented MESO based on Hugging Face Transformers (v4.47.1) [25], integrating the three key optimizations described in section III-B.

Models. We evaluated our framework using the open-source Mixtral-8x7B [7] model, which contains a total of 46.7 billion parameters. For speculative decoding, we employed Mistral-7B [26] (a 7B dense model) as the draft model. All model weights were loaded in bfloat16 format by default. MESO is designed to be broadly compatible with various Transformer-based architectures.

Hardware. We evaluated MESO on the hardware configuration detailed in Table II. This setup represents a typical resource-constrained environment where the GPU memory (*e.g.*, 40GB) is significantly smaller than the model size ($\approx$ 87GB in bfloat16), mandating offloading. We utilized the A100 (40GB) platform primarily to ensure fair reproducibility, as baselines such as SpecOffload require over 18GB of peak GPU memory (as shown in Fig. 3 and Fig. 4) and would fail due to Out-of-Memory (OOM) errors on GPUs with smaller capacities (*e.g.*, 12GB or 16GB). By using a 40GB GPU, we can fully capture the performance characteristics of all baselines while demonstrating that MESO is capable of operating within a much tighter memory budget.

Datasets. We selected widely adopted LLM benchmarks featuring diverse prompt length distributions and task types to ensure a comprehensive evaluation, as summarized in Table III).

Baselines. Comparative analysis is performed against four baselines, all designed to mitigate GPU memory constraints:

(1) Hugging Face Accelerate [30] (version 1.5.2, herein denoted as *Accelerate*), which facilitates weight offloading for designated layers based on device mapping;
(2) DeepSpeed Zero-Inference [16] (version 0.16.1, herein denoted as *DeepSpeed*), supporting comprehensive offloading of the entire model weights to CPU or disk;
(3) Fiddler [8], which strategically leverages heterogeneous CPU and GPU resources to optimize inference for MoE models;
(4) SpecOffload [9], the foundational approach embedding SD within an offloading paradigm where the draft model persistently resides in GPU memory.

Notably, all evaluated baseline systems provide native compatibility with the Mixtral architecture.

Parameters Configuration. We focused on evaluating workloads with batch sizes of 16, 32, and 64. To assess scalability, we also varied the generation length, denoted as N_t, ranging from 16 to 256 tokens. Such medium-to-high concurrency requests naturally arise in multi-input scenarios, such as industrial settings processing visual captioning requests from over 20 cameras simultaneously, or community AI gateways aggregating and batch-processing queries from multiple households.

Metrics. We report the following metrics:

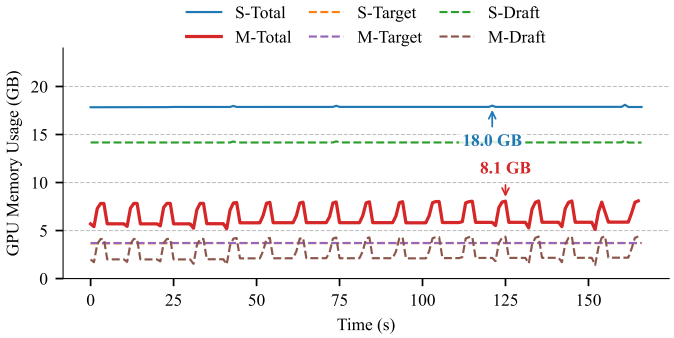

Fig. 3. Dynamic comparison of GPU memory usage between MESO and SpecOffload during inference. **S**: SpecOffload, **M**: MESO. The results of their target models overlap.

(1) Throughput (token/s): Reflects inference speed, measured in tokens per second.
(2) Peak GPU Memory Usage (GB): Reflects resource consumption.
(3) Memory Efficiency (token/s/GB): The core metric, this measures the cost-effectiveness of resource utilization, as defined in (1).

B. Main Results

Memory Efficiency: As shown in Table I, MESO achieves the highest memory efficiency across all test scenarios. Specifically, regarding the improvement in memory efficiency relative to the best baseline in each scenario, MESO achieves 1.02×-1.83× the efficiency on the HumanEval dataset. On the more challenging SummEval dataset, this advantage further expands to 1.0×-2.25×. Averaged across all scenarios, the improvement is **1.35**×, with a peak observed gain of **2.25**×. This demonstrates that MESO effectively adapts to stricter memory constraints, realizing the highest throughput gain per unit of memory occupancy.

Resource Usage and Absolute Performance:

(1) When inferring the 46.7B model, MESO maintains a peak GPU memory usage of only 8GB. Compared to SpecOffload (18GB), peak GPU memory usage is reduced by over **50%** (as shown in Fig. 3 and Fig. 4). This reduction fundamentally expands the deployment feasibility window: while SpecOffload exceeds the memory capacity of typical high-end consumer GPUs (*e.g.*, those with 12GB or 16GB GPU memory), MESO operates comfortably within these limits.
(2) Under identical memory constraints, MESO achieves a throughput **2.7**× higher than the second-best solution, achieving the optimal balance between resource consumption and inference speed, as shown in Fig. 5.

Throughput Scalability with Generation Length: We further analyzed throughput trends across varying generation lengths (N_t) with a batch size of 32, as shown in Fig. 6. Standard offloading methods (*e.g.*, DeepSpeed and Fiddler) suffer from performance stagnation; for instance, Fiddler's throughput hovers around 3.5 token/s regardless of sequence

TABLE I
COMPREHENSIVE COMPARISON OF MEMORY EFFICIENCY (η) (TOKEN/S/GB) ON THE MIXTRAL-8X7B MODEL ACROSS MULTIPLE DATASETS, BATCH SIZES, AND GENERATION LENGTHS.

Dataset	Method	Generation Length (tokens)														
		Batch Size = 16					Batch Size = 32					Batch Size = 64				
		16	32	64	128	256	16	32	64	128	256	16	32	64	128	256
HumanEval	Accelerate	0.039	0.039	0.039	0.037	0.035	0.033	0.033	0.033	0.032	0.029	0.023	0.023	0.022	0.021	0.018
	DeepSpeed	0.192	0.192	0.192	0.191	0.192	0.384	0.382	0.386	0.365	0.320	0.467	0.467	0.467	0.468	0.404
	Fiddler	0.497	0.525	0.545	0.579	0.581	0.544	0.535	0.549	0.558	0.523	0.555	0.557	0.568	0.601	0.613
	SpecOffload	0.199	0.248	0.287	0.327	0.358	0.465	0.571	0.606	0.656	0.632	0.954	1.118	1.160	1.180	1.245
	MESO (Ours)	**0.509**	**0.574**	**0.628**	**0.631**	**0.669**	**0.907**	**1.046**	**1.089**	**1.131**	**0.932**	**1.188**	**1.352**	**1.281**	**1.418**	**1.372**
SummEval	Accelerate	0.028	0.030	0.030	0.029	0.028	0.019	0.020	0.020	0.020	0.020	0.012	0.013	0.013	0.013	0.013
	DeepSpeed	0.154	0.155	0.151	0.151	0.151	0.217	0.210	0.211	0.210	0.211	0.125	0.126	0.131	0.125	0.114
	Fiddler	0.311	0.375	0.377	0.387	0.397	0.297	0.287	0.280	0.290	0.293	0.286	0.292	0.293	0.303	0.306
	SpecOffload	0.166	0.195	0.202	0.212	0.309	0.292	0.385	0.437	0.435	0.575	0.590	0.562	0.623	0.899	1.207
	MESO (Ours)	**0.346**	**0.382**	**0.385**	**0.413**	**0.508**	**0.669**	**0.736**	**0.740**	**0.836**	**0.989**	**0.927**	**0.876**	**0.905**	**0.981**	**1.240**
Samsum	Accelerate	0.032	0.030	0.025	0.021	0.019	0.016	0.015	0.016	0.015	0.014	0.010	0.004	0.007	0.014	0.015
	DeepSpeed	0.166	0.168	0.168	0.168	0.162	0.231	0.231	0.231	0.232	0.229	0.121	0.120	0.122	0.121	0.114
	Fiddler	0.301	0.354	0.413	0.423	0.399	0.370	0.375	0.382	0.395	0.402	0.324	0.336	0.339	0.336	0.325
	SpecOffload	0.143	0.193	0.231	0.301	0.355	0.266	0.381	0.427	0.560	0.612	0.542	0.727	0.868	1.088	1.263
	MESO (Ours)	**0.313**	**0.392**	**0.472**	**0.569**	**0.671**	**0.563**	**0.727**	**0.761**	**0.989**	**1.160**	**0.734**	**0.866**	**1.007**	**1.215**	**1.385**

TABLE II
HARDWARE CONFIGURATION.

Component	Specification
GPU	A100-SXM4
VRAM	40GB
PCIe	Gen4 x 16
CPU	Xeon Platinum 8378A
DRAM	1TB

TABLE III
DATASETS.

DataSet	Task	Sizes
HumanEval [27]	Coding	164 problems
Summeval [28]	Summarization	100 articles
Samsum [29]	Summarization	$16k$ dialogues

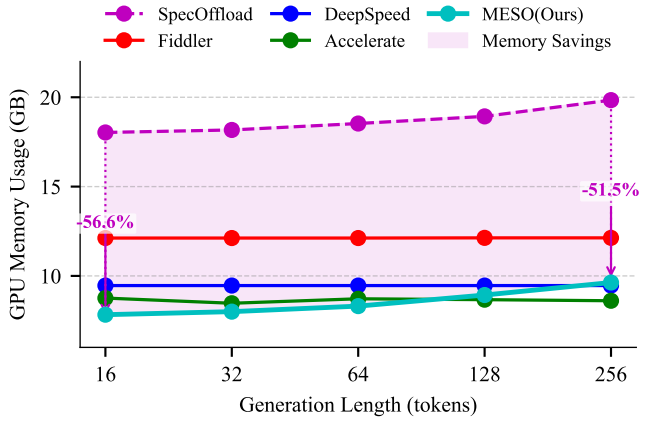

Fig. 4. Impact of generation length on peak GPU memory usage. Compared to SpecOffload, MESO drastically reduces memory consumption by over 50% (shaded region), maintaining a low usage ($\approx$ 8GB) comparable to pure offloading baselines.

length. This indicates that their synchronous data transfer overheads dominate the inference cycle, negating the benefits of increased computational density. *Conversely*, MESO demonstrates superior scalability. As N_t extends from 16 to 256, our throughput surges from 5.24 to 9.52 token/s, representing an **81.7%** gain. This trajectory mirrors the draft-resident baseline (SpecOffload), confirming that our Attention Computation Delegation and Asynchronous Pipeline successfully mask I/O latency, thereby validating the framework's robustness for long-context tasks on resource-constrained devices.

C. Ablation Study and Analysis

We conducted a progressive ablation study to quantitatively analyze the contributions of the optimization components. The results are summarized in Table IV.

First, when only Draft Model Offloading (MO) is enabled, memory efficiency improves by **22.78%**. This is primarily attributed to the substantial reduction in GPU memory usage resulting from offloading the draft model.

Subsequently, introducing Load-to-GPU Pinning (LGP) optimizes data transmission between the host and device. By enabling Pinned Memory for high-frequency components, we achieve a **13.26%** increase in throughput and elevate the memory efficiency gain to **37.34%**.

Finally, integrating Fine-grained Component Loading (FCL)

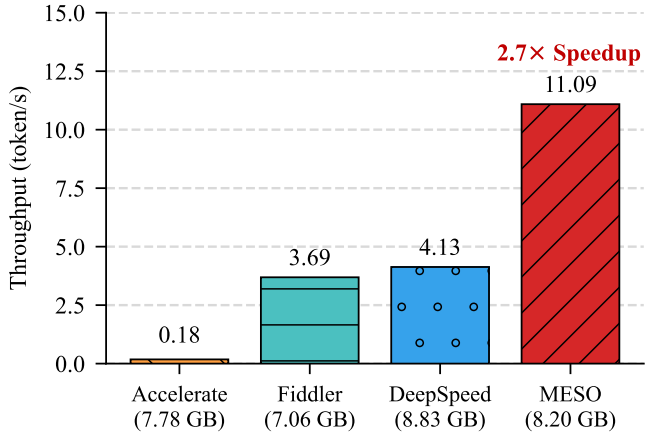

Fig. 5. Throughput comparison under strict memory constraints (< 9GB). Under conditions where standard SpecOffload exceeds memory limits, MESO achieves 11.09 token/s, delivering a 2.7× speedup over DeepSpeed and significantly outperforming Fiddler.

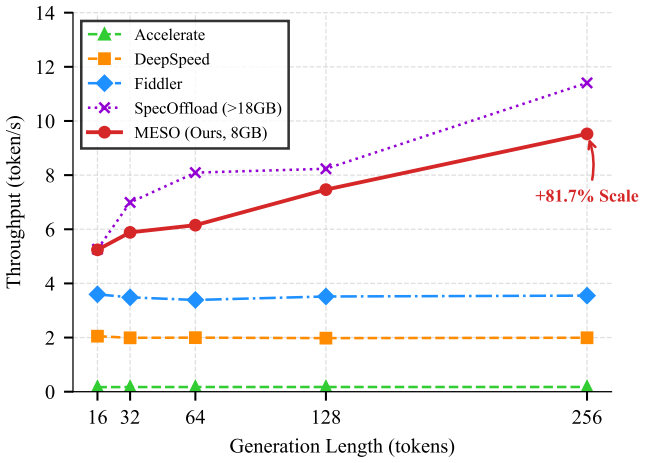

Fig. 6. Throughput scalability across different generation lengths (N_t). MESO demonstrates significant performance scaling, whereas baselines remain stagnant.

effectively transforms synchronous blocking loading into an asynchronous parallel process. This optimization further boosts the total throughput improvement to **21.57%** and maximizes the memory efficiency gain to **47.47%**.

D. Quantization Synergy Verification

To explore the limits of memory efficiency, we integrated GPTQ quantization into MESO, creating a multi-dimensional optimization space. Table V presents the performance of three representative configurations, revealing a distinct trade-off spectrum:

(1) *Baseline (MESO):* Operates at full precision (BF16), prioritizing generation quality. While it incurs higher memory usage (8.08 GB), it serves as a robust baseline for accuracy-sensitive tasks.

(2) *Ultra Mode (Double Quantization):* By quantizing both the draft and target models to INT4, we achieve an extreme memory usage of 4.75 GB. Although the throughput decreases to 4.91 token/s due to the reduced acceptance rate of the quantized draft model, this mode is critical for enabling 46.7B model inference on entry-level edge devices where deployment was previously impossible.

(3) *Medium Mode (Target-Only Quantization):* This configuration quantizes only the target model while keeping the draft model in high precision. Interestingly, this yields the highest throughput (10.632 token/s) and peak memory efficiency (1.65 token/s/GB). This is because the high-precision draft model maintains a high acceptance rate, while the quantized target model significantly reduces the I/O transfer overhead during verification.

These results empirically validate the orthogonality between static quantization and dynamic offloading. The success of the *Medium Mode* specifically highlights a novel optimization strategy for offloading-based SD systems: "Quantize the heavy target for bandwidth, keep the light draft precise for acceptance." This hybrid approach maximizes the system's overall efficiency, offering a versatile solution that can be tailored to varying hardware constraints.

TABLE IV
ABLATION STUDY ON THE HUMANEVAL DATASET.

	Memory efficiency (token/s/GB)	**Throughput** (token/s)
No optimizations	0.632	-
MO	0.776(+ 22.78%)	7.632
MO + LGP	0.868(+ 37.34%)	8.644(+ 13.26%)
MO + LGP + FCL	**0.932**(+ 47.47%)	**9.278**(+ 21.57%)

Note: **MO** = Draft Model Offloading, **LGP** = Load-to-GPU Pinning, **FCL** = Fine-grained Component Loading. Experimental configuration: prefill batch size = 8, decoding batch size = 32, draft batch size = 16, draft max new tokens = 8, output sizes = 256. Gray numbers indicate percentage increase from the first value in the same column.

TABLE V
SYNERGY TEST RESULTS OF QUANTIZATION (GPTQ) ON HUMANEVAL DATASET.

	Draft Q.	**Target Q.**	**Memory Usage (GB)**	**Throughput (token/s)**	**ME (token/s/GB)**
MESO	-	-	8.08	<u>8.449</u>	<u>1.05</u>
Ultra	√	√	**4.75**	4.910	1.03
Medium	-	√	<u>6.43</u>	**10.632**	**1.65**

Note: **Q.** = Quantization, **ME** = Memory Efficiency. "√" indicates enabled, "-" indicates disabled. **Bold** indicates the best performance or lowest GPU memory usage. <u>Underline</u> indicates the second-best results. Decode batch size = 32.

V. CONCLUSION

We propose the MESO to address the critical issue of low memory efficiency in existing LLM inference frameworks when deployed in resource-constrained devices. While reducing peak GPU memory usage by over 50%, MESO effectively mitigates the negative impact of offloading-induced I/O latency

on throughput through fine-grained component asynchronous loading and pinned memory optimization. Experimental results demonstrate that MESO achieves superior memory efficiency across diverse datasets and experimental settings, significantly outperforming existing baselines. Beyond providing a practical and efficient inference solution, this work systematically defines and optimizes "Memory Efficiency" as a core evaluation metric, establishing new directions and standards for future research on LLM deployment in resource-constrained devices.

ACKNOWLEDGMENT

This work was supported by the National Key Research and Development Program under Grant 2024YFB4506200, the Science and Technology Innovation Program of Hunan Province under Grant 2024RC1048, the Science and Technology Innovation Program of Hunan Province under Grant 2025RC3121.

REFERENCES

[1] Rongjie Yi, Liwei Guo, Shiyun Wei, Ao Zhou, Shangguang Wang, and Mengwei Xu, "Edgemoe: Empowering sparse large language models on mobile devices," *IEEE Transactions on Mobile Computing*, 2025.

[2] Zhilong Liu, Long Peng, Wenzhu Wang, Ke Li, Binrui Zeng, Jie Yu, et al., "Accelerating llm inference on risc-v edge devices via vector extension optimization," in *International Conference on Intelligent Computing*. Springer, 2025, pp. 515–526.

[3] Hao Xu, Long Peng, Shezheng Song, Xiaodong Liu, Ma Jun, Shasha Li, et al., "Camel: Energy-aware llm inference on resource-constrained devices," *arXiv preprint arXiv:2508.09173*, 2025.

[4] Siyuan Mu and Sen Lin, "A comprehensive survey of mixture-of-experts: Algorithms, theory, and applications," *arXiv preprint arXiv:2503.07137*, 2025.

[5] Andrii Skliar, Ties van Rozendaal, Romain Lepert, Todor Boinovski, Mart Van Baalen, Markus Nagel, et al., "Mixture of cache-conditional experts for efficient mobile device inference," *Transactions on Machine Learning Research*, 2025.

[6] Peng Tang, Jiacheng Liu, Xiaofeng Hou, Yifei Pu, Jing Wang, Pheng-Ann Heng, et al., "Hobbit: A mixed precision expert offloading system for fast moe inference," *arXiv preprint arXiv:2411.01433*, 2024.

[7] Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, et al., "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[8] Keisuke Kamahori, Tian Tang, Yile Gu, Kan Zhu, and Baris Kasikci, "Fiddler: CPU-GPU orchestration for fast inference of mixture-of-experts models," in *The Thirteenth International Conference on Learning Representations*, 2025.

[9] Xiangwen Zhuge, Xu Shen, Zeyu Wang, Fan Dang, Xuan Ding, Danyang Li, et al., "Specoffload: Unlocking latent gpu capacity for llm inference on resource-constrained devices," *arXiv preprint arXiv:2505.10259*, 2025.

[10] Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, et al., "Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding," *arXiv preprint arXiv:2401.07851*, 2024.

[11] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, et al., "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th symposium on operating systems principles*, 2023, pp. 611–626.

[12] Weilin Cai, Juyong Jiang, Fan Wang, Jing Tang, Sunghun Kim, and Jiayi Huang, "A survey on mixture of experts in large language models," *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[13] Yaniv Leviathan, Matan Kalman, and Yossi Matias, "Fast inference from transformers via speculative decoding," in *Proceedings of the 40th International Conference on Machine Learning*, 2023, ICML.

[14] Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, et al., "Medusa: Simple llm inference acceleration framework with multiple decoding heads," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, ICML.

[15] Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang, "Eagle: speculative sampling requires rethinking feature uncertainty," in *Proceedings of the 41st International Conference on Machine Learning*, 2024, ICML.

[16] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, et al., "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[17] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, et al., "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31094–31116.

[18] Zhiyuan Fang, Yuegui Huang, Zicong Hong, Yufeng Lyu, Wuhui Chen, Yue Yu, et al., "Klotski: Efficient mixture-of-expert inference via expert-aware multi-batch pipeline," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2025, vol. II, pp. 574–588.

[19] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*. 2024, SOSP, pp. 590–606, Association for Computing Machinery.

[20] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[21] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, et al., "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of machine learning and systems*, vol. 6, pp. 87–100, 2024.

[22] Ren Chen and Viktor K. Prasanna, "Computer generation of high throughput and memory efficient sorting designs on fpga," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 11, pp. 3100–3113, 2017.

[23] Daon Park and Bernhard Egger, "Improving throughput-oriented llm inference with cpu computations," in *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*. 2024, PACT, pp. 233–245, Association for Computing Machinery.

[24] Shiyi Cao, Shu Liu, Tyler Griggs, Peter Schafhalter, Xiaoxuan Liu, Ying Sheng, et al., "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*. 2025, vol. I of *ASPLOS*, pp. 715–730, Association for Computing Machinery.

[25] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, et al., "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[26] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, et al., "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[27] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, et al., "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[28] Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev, "Summeval: Re-evaluating summarization evaluation," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 391–409, 2021.

[29] Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer, "Samsum corpus: A human-annotated dialogue dataset for abstractive summarization," in *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, 2019, pp. 70–79.

[30] Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, et al., "Accelerate: Training and inference at scale made simple, efficient and adaptable," *[Computer software]. Hugging Face. https://github. com/huggingface/accelerate*, 2022.