# LLM-Mesh: Enabling Elastic Sharing for Serverless LLM Inference

Chuhao Xu
Shanghai Jiao Tong University
Shanghai, China

Zijun Li
Shanghai Jiao Tong University
Shanghai, China

Quan Chen
Shanghai Jiao Tong University
Shanghai, China

Han Zhao
Shanghai Jiao Tong University
Shanghai, China

Minyi Guo
Shanghai Jiao Tong University
Shanghai, China



## Abstract

The rise of LLMs has driven demand for private serverless deployments, characterized by moderate-scale models and infrequent requests. While existing solutions follow exclusive GPU deployment, we take a step back to explore modern platforms and find that: Emerging CPU architectures with built-in accelerators are capable of serving LLMs but remain underutilized, and both CPUs and GPUs can accommodate multiple LLMs simultaneously.

We propose LLM-Mesh, a serverless inference scheme for small-to-mid-sized LLMs that enables elastic sharing across heterogeneous hardware. LLM-Mesh tackles three fundamental challenges: (1) precise, fine-grained compute resource allocation at token-level to handle fluctuating computational demands; (2) a coordinated and forward-looking memory scaling mechanism to detect out-of-memory hazards and reduce operational overhead; and (3) a dual approach that reduces resource fragmentation through proactive preemption and reactive bin-packing. Experimental results on 4 32-core CPUs and 4 A100 GPUs show that LLM-Mesh improves service capacity by 44% - 63% through sharing, while further leveraging CPUs boosts this to 91% - 159%.

## 1 Introduction

Large Language Models (LLMs) have seen widespread adoption, with many providers (e.g., OpenAI [4], Anthropic [10]). Driven by the need for customization and privacy, individuals and medium-sized enterprises are increasingly seeking to deploy private models with a serverless approach [5–7], offloading infrastructure management to clouds. A closer examination of this deployment reveals two key characteristics that closely align with the typical patterns of serverless [27, 49] workloads: (1) small to mid-sized models make up the majority [3], and (2) model invocations tend to be infrequent and highly variable [21, 64].

Considering high computational demands and strict SLO requirements [13, 65], serverless solutions [21, 34, 61] often allocate dedicated GPU resources to accommodate general workloads, especially large-scale LLMs (e.g., over 70B) deployment considerations. However, above mechanism fundamentally mismatches with the scenario where numerous small-sized models are infrequently invoked with serverless workload patterns. The key issue lies in the scarcity of GPUs relative to the number of models, while the resource over-provisioning makes each model occupy an entire GPU.

Through systematic investigations of modern GPU cloud platforms, we re-examine the deployment characteristics for small-to-medium-scale LLMs, revealing two key opportunities. First, clusters have abundant idle CPU resources, and utilizing their built-in accelerators (e.g., Intel AMX [12, 39]) can independently support small to medium-sized models while meeting production-grade SLOs. Second, given the low-frequency, serverless-like workload pattern, individual models usually do not fully saturate resources on CPUs/GPUs, making the colocation of multiple LLMs practical.

We therefore are motivated to design a serverless LLM serving scheme that prioritizes heterogeneous hardware-agnostic resource allocation, and automatically allocates and scales heterogeneous resources on-demand, enabling elastic provisioning. However, the heterogeneous variation patterns in compute and memory demands present three fundamental design challenges.

First, for LLM inference, the computational demand fluctuates significantly during token generation, especially as the first token of each request undergoes the prefill stage. To improve resource efficiency, it is needed to perform precise demand quantification, allocation and budget at token-level. The scheme should make fine-grained compute resource allocation at token-level, while generalizing such compute management mechanism to heterogeneous environments.

Second, diverse workloads result in volatile key-value cache demands, while dynamical adjustments can introduce non-trivial operational overheads, necessitating an overhead-aware memory scaling strategy. Meanwhile, asynchronous and sensitive memory operations should also be globally orchestrated to avoid out-of-memory issues. To this end, a global-coordinated and overhead-aware memory scaling strategy is required to perform memory allocation effectively.

Third, a crowded sharing environment limits the scale-up potential of a single instance, leading to multiple fragmented instances of the same model across heterogeneous environments. Batch-unaware scheduling would create resource fragmentation, substantially degrading resource utilization.

Defragmentation mechanisms are essential to maintain sustained resource efficiency in serverless environments.

Resolving above three challenges, we propose **LLM-Mesh**, a Serverless LLM Inference scheme achieving the elastic deployment for small-to-medium-scale LLMs. LLM-Mesh abstracts heterogeneous hardware into CPU/GPU nodes, decoupling resource management through compute and memory subsystems. The compute subsystem, driven by request headroom, efficiently budgets and schedules instances by quantification-based shadow validation and real-time token-level orchestration. For memory subsystem, it performs predictive memory demands via watermark considering the tradeoff of scaling overhead. To avoid OOM hazards, it orchestrates multiple memory adjustments in a controlled and parallel manner, inspired by Tomasulo's algorithm. Lastly, to maintain runtime efficiency, LLM-Mesh introduces a dual defragementation approach: proactively improving per-instance efficiency through preemption while simultaneously employing a bin-packing strategy to eliminate fragmentation.

The main contributions of this paper are as follows.

- **Systematic investigation of LLM serving on heterogeneous resource.** The identified CPU/GPU sharing opportunities motivate a elastic-provisioning design.
- **Principles for sharing small-to-mid-sized LLMs under serverless paradigm.** Based on investigation, we construct guidelines considering unique characteristics of heterogeneous hardware and inference procedure.
- **A resource management system with unified hardware abstraction.** Based on LLM-Mesh, we implement two subsystems that transparently manage hardware while ensuring efficient and precise resource sharing.

We evaluate LLM-Mesh with real-world LLM datasets [43] and serverless workloads [49]. Experimental results on 4 32-core CPU nodes and 4 A100-80GB GPU nodes demonstrate that LLM-Mesh improves serving capacity by 44% - 63% through elastic sharing, and leveraging CPU resource further boosts this improvement to 91% - 159%.

## 2 Background and Motivation

### 2.1 LLM Inference Process

For LLM inference, users submit requests containing input text, which the inference engine processes iteratively [13, 36, 51, 56]. Each iteration produces a new token that streams to users in real-time through incremental response generation.

A request progresses through two distinct stages [42, 43, 65]. The initial prefill stage occurs during the first iteration, where the engine constructs the key-value (KV) cache [11, 29] from input tokens and generates the first output token. For subsequent decoding, the engine appends newly generated tokens to KV-cache while producing one token per iteration until generation completes. Modern inference engines also employ continuous batching [59], which enables dynamic incorporation of new requests into active processing batches, avoiding dedicated instance creation per request.

Interactive-based LLM serving systems should follow strict Service Level Objectives (SLOs) for user requests. Two critical performance metrics focus on these objectives: **1. Time to First Token (TTFT)** – how early the response stream begins after a user submits a request, and **2. Time per Output Token (TPOT)** – how quickly the response stream generates. TTFT requirements typically constrain initial response latency to a few seconds [65], considering for long input sequences. TPOT aims to return output text faster than the human reading speed, which is around 250 tokens/min [13].

### 2.2 Small-to-Medium-Scale LLMs and Cloud Hosting

Small-to-medium-scale LLMs (e.g., 7B, 13B parameters) have demonstrated capability in addressing most application scenarios while achieving substantially lower operational costs compared to larger parameters variants [2, 45]. Concurrently, increasing demands for customization, coupled with data privacy and security requirements, have driven more developers to adopt private, self-managed LLM deployments. As a result, thousands of variants have emerged-just within the LLaMA-2-7B series [1]. An examination of these hosted models in HuggingFace reveals two significant characteristics that exhibit inherent features of serverless workloads [49, 63].

- First, small-to-medium-scale models dominate in private deployment scenarios. Data from HuggingFace [3] indicates that models with fewer than 8 billion parameters constitute 60% of user preference and 87% of total downloads.
- Second, invocation patterns exhibit low frequency and high variability [21, 64]. Private models, often serving limited user groups, experience irregular access patterns characterized by randomness, burstiness, and diurnal fluctuations.

For small-to-medium-scale LLMs, serverless LLM serving offers an appealing cloud-hosted solution [5–7]. With serverless paradigm, developers can focus on defining inference SLOs while offloading infrastructure management to clouds. Serverless providers allocate resources as workload changes, enabling on-demand scaling without fixed provisioning.

### 2.3 Problems with Existing Serving Mechanisms

Given the high computational demands of LLM inference and the strict two-stage SLO requirements, existing serverless solutions heavily rely on powerful GPUs, following an exclusive GPU resource provisioning mechanism. This mechanism, considering the compatibility and efficiency for large-scale LLMs and high-throughput environments, performs well in serving giant LLMs (e.g., LLaMA-2-70B). However,

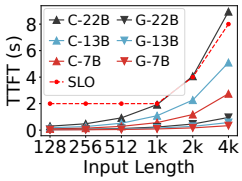



**Figure 1.** The TTFT metric of diverse models using CPU/GPU.

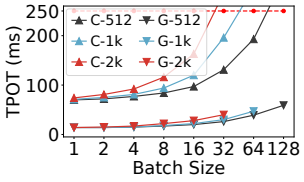



**Figure 2.** The TPOT metric under different sentence length of Llama-2-7B.

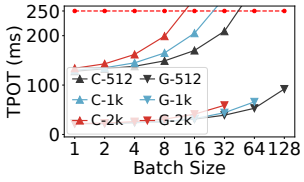



**Figure 3.** The TPOT metric under different sentence length of Llama-2-13B.

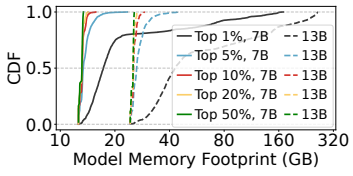



**Figure 4.** The memory footprint of different models under real-world workloads.

we find it fundamentally mismatched for small-to-medium-scale LLMs due to resource over-provisioning. The requirement for dedicated GPU allocation per model, even for small-scale LLMs, forces concurrent requests across models heavily queuing for awaiting GPU availability.

The ill-suited fact of existing serving mechanisms motivates us to take a step back and reassess the evolving hardware architectures and the practical workload scales of small-to-medium-sized LLMs. Instead of being constrained by scarce GPU resources, alternative hardware platforms might offer viable solutions. Moreover, these heterogeneous resources could potentially enable efficient multi-model sharing, rather than being exclusively dedicated to a single model. Therefore, we next conduct a systematic investigation of the heterogeneous architecture to explore the opportunities of serverless deployment for small-to-medium-scale LLMs.

## 3 Heterogeneous Resource Sharing

Modern data centers are inherently heterogeneous [15, 38, 55]. Even a GPU cluster is equipped with CPU nodes for preprocessing tasks. Given the report of low CPU utilization in GPU nodes [25, 26] and the emerging CPU architecture [39], it is also worth exploring the potential of idle CPU resource.

### 3.1 CPU Sharing Opportunity

Despite the presence of spare CPU resources, it is still uncertain whether they can be used for LLM inference, considering the stringent SLO requirements and the heavy computational load. However, it is worth noting that recent CPU architectures have incorporated specialized components to meet the proliferation of AI workloads. Starting with the 4th Gen Intel Xeon, Intel introduced the Advanced Matrix Extensions (AMX) [12, 39], a dedicated hardware block designed to accelerate matrix operations. Therefore, CPUs may now be capable of LLM inference with such architectural innovations.

To reveal the uncertainty, we replaced the default GPU backend of vLLM with OpenVINO [8], the state-of-the-art for CPU inference. We run vLLM+OpenVINO on a 32-core Intel Xeon 6462C@3.3 GHz CPU, and collect the TTFT and TPOT metrics of three different sized models (Llama-2-7B, Llama-2-13B, and Codestral-22B) under various sentence lengths and batch sizes. As discussed in §2.1, following previous works [13, 65], we use the 16-bit precision and set the TTFT SLO to $\max(2, \text{input\_length}/512)$ s and the TPOT SLO to 0.25 s.

Figure 1 demonstrates the TTFT performance, corresponding to the elapsed time of the prefill stage, which is known for being highly compute-intensive [43]. In this figure, "C-7B" represents using CPU with Llama-2-7B. We compare the CPU results with GPU (one A100) and SLO requirements. Although the CPU is much slower than the GPU, it surprisingly achieved the SLO on both the 7B and 13B models. However, it failed on the larger 22B model as the input length increased, due to its limited computation capability.

We further investigate the TPOT performance of the 7B and 13B models, representing the performance of the decode stage, as shown in Figure 2 and Figure 3, respectively. In the figure, "C-512" represents using CPU with 512 sentence length. Unlike the prefill stage, the compute cost of the decode stage for each request can be amortized through batching, as long as there is sufficient parallel computing capacity. We find that the CPU not only meets TPOT SLO with ease, but can also utilize batching to improve throughput, just like the GPU. For instance, when using CPU to serve the 7B model with 1k sentence length, the TPOT of a 16-batch only increases by 69% compared to 1-batch. However, as the batch size continues to increase, the CPU's TPOT starts to grow significantly and exceeds the SLO, while the GPU's TPOT remains relatively short. We also find that the TPOT is correlated with sentence length. For example, when serving the 13B model on CPU with a 32-batch, there is a 2.0X difference in TPOT between 512-length and 2k-length, and the latter already violates the SLO.

**Takeaway.** CPU presents an opportunity for resource sharing with architectural support however remains constrained in LLM scenarios. As a compute-bound resource, CPU performance exhibits strong sensitivity to different inference stages as well as sequence length and batch size variations, leading to frequent SLO violations.

### 3.2 GPU Sharing Opportunity

A GPU can be shared by multiple LLM models when the global memory is large enough to host the data.

**Memory Footprints.** The memory footprint of a model instance mainly consists of model parameters and the KV-cache of requests. While the parameters part is fixed, the KV-cache part is dynamic with the request concurrency and length of output. To investigate memory footprint under real-world workloads, we set the length of each request sampled from Azure LLM Trace [43]. Since it does not contain multi-LLM invocation patterns, following ServerlessLLM [21], we fire requests based on Azure Serverless Trace [49].

Figure 4 demonstrates the results of the 7B and 13B model under real-world workloads on 4 A100-80GB GPUs. In this figure, "Top 1% (7B)" represents mapping the Llama-2-7B model to the 1st percentile function in the Azure Serverless Trace sorted by request counts. Since each instance occupies one GPU, when the memory footprint exceeds 80GB, it indicates that two or more instances have been created.

First, for the 7B and 13B models, they need at least 14GB and 26GB of memory, respectively, corresponding to the memory footprint of the model parameters, regardless of the workload. Selecting a smaller percentile function would correspond to a higher request load, i.e., a larger memory footprint due to increased KV-cache usage. When selecting the top 1% function, the memory footprint reaches up to 169GB and 263GB for the 7B and 13B models, respectively. These observations demonstrate that under peak load conditions, LLMs require exclusive access to multiple dedicated GPUs. Notably, analysis of the top 1% most memory-intensive workloads reveals that during 50% of operational periods, memory consumption remains below 17GB for 7B parameter models and 43GB for 13B parameter configurations.

**Takeaway.** The memory footprint of one model fluctuates with the workload but remains low in most cases. Considering that GPUs like A100 are equipped with 80GB of memory, a single model often does not require the full capacity. However, the memory footprint varies quickly as the tokens are continuously generated, requiring careful planning to achieve efficient memory sharing. GPUs, as memory-bound hardware resources at the architectural level, require careful resource allocation planning to enable resource sharing for serverless LLMs without heavy computational demands.

# 4 Design Overview of LLM-Mesh

Taking advantage of the above opportunities, we present *LLM-Mesh*, a Serverless LLM Inference scheme designed for small-to-medium-scale LLMs in heterogeneous datacenters. It enables the elastic sharing via the resource-type- and resource-demand-friendly deployment. The resource-type-friendly deployment prioritizes heterogeneous hardware-agnostic resource allocation, avoiding tight coupling to scarce GPU resource. Demand-aware resource management enables elastic sharing of underlying hardware instead of over-provisioning.

LLM-Mesh manages GPU and CPU nodes as heterogeneous nodes through the compute and memory subsystems. When a new request arrives, LLM-Mesh prioritizes CPU nodes when attempting to assign it to candidate instances. The compute subsystem performs shadow validation by evaluating the availability of request headroom-based token-level scheduling, while the memory subsystem verifies real-time memory capacity against node utilization. Following successful validation by both subsystems, the request is dispatched to the selected candidate. Subsequently, the compute-subsystem orchestrates execution at token-level, focusing on request headroom (see §5). The memory-subsystem will employ a watermark-based KV-cache auto-scaling and a hazard-aware out-of-order memory operarion strategy (see §6).

If all candidate instances fail validation, LLM-Mesh introduces a proactive defragmentation mechanism, which attempts to preempt resources from neighboring instances to fit the request into an existing one, thereby avoiding fragmentation (see §7). Upon request completion, LLM-Mesh does not require any specific actions other than scaling down KV-cache for the corresponding instance by the memory-subsystem. Additionally, following a keep-alive policy, LLM-Mesh recycles an instance if it remains idle until timeout without handling any requests.

# 5 Headroom-Driven Compute Subsystem

## 5.1 Headroom-based Token-level Scheduling

To allocate computational resource at token-level, since each iteration ends up with a new token, LLM-Mesh dynamically orchestrates the compute iterations of multiple instances. Specifically, as illustrated in Figure 5, it selects one instance at a time to compute one iteration and generate a batch of tokens. Once complete, it moves on to the next instance for another iteration cycle, and the process repeats.

By continuously assigning token-level tasks to instances, the node is full-time utilized without idle periods. However, it is still uncertain which instance should be selected for each scheduling cycle. To minimize SLO violations, LLM-Mesh prioritizes selecting the instance handling the most urgent request.

LLM-Mesh introduces *headroom* to characterize the degree of urgency. Let $TTFT_{SLO}$ and $TPOT_{SLO}$ represent the SLO for TTFT and TPOT, respectively. Suppose a request starts at time $ST$, has already generated $O$ tokens, and the current time is $CT$. The headroom for this request, which represents the maximal delay time for generating the next token within the SLO, is given by:

$$headroom = ST + TTFT_{SLO} + TPOT_{SLO} \cdot O - CT \quad (1)$$

Therefore, at each scheduling cycle, LLM-Mesh selects the instance with the shortest request headroom and assigns it an iteration. In Figure 5, it first selects instance-2. Suppose the

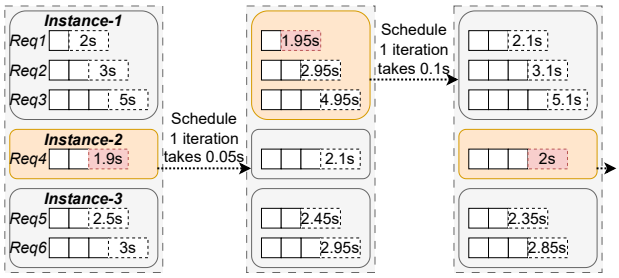



**Figure 5.** Procedure of token-level scheduling. At each cycle, LLM-Mesh schedules the instance with the shortest *headroom*.

$TPOT_{SLO}$ is 0.25 s and the iteration takes 0.05 s, the headroom then updates to $1.9 - 0.05 + 0.25 = 2.1$ s. LLM-Mesh then re-compares the headroom and repeats the process.

### 5.2 Performance Quantification

Since headroom represents the time a request can delay its output, a negative headroom indicates that an SLO violation has occurred. To make sure this does not happen, it is essential to quantify the performance of each model instance, specifically the computation time per iteration under varying loads. Since a prefill iteration is significantly different from a decode iteration, LLM-Mesh characterizes them separately.

**Quantify Prefill Time.** As shown in Figure 1, the prefill time is approximately linearly correlated with the input sentence length. Therefore, LLM-Mesh uses linear interpolation. For a given model, LLM-Mesh collects the TTFT results for an input sentence length samples $S_L$. Then, for a new request of input sentence length $L$, it finds the two closest known points and applies the interpolation.

**Quantify Decode Time.** As shown in Figure 2 and 3, the time of decode iteration is correlated with both length and batch size. Thus, LLM-Mesh uses these two factors as two dimensions and applies 2D linear interpolation. For a given model, LLM-Mesh generates the batch size samples $S_B$ and the average sentence length samples $S_L$. For each $B' \in S_B$ and $L' \in S_L$, LLM-Mesh collects the corresponding TPOT result. Then, for a batch size $B$ and average sentence length $L$, it finds the four closest known points and applies the interpolation.

Considering the characteristics of heterogeneous hardware, LLM-Mesh quantifies on CPU and GPU respectively. To reduce sampling overhead, it uses $2^X$ to generate the samples of $S_B$ and $S_L$. If a model's maximum sequence length is $L_{\max}$ (e.g., 4096) and the maximum batch size is $B_{\max}$ (e.g., 256), LLM-Mesh only needs to collect $O(\log_{L_{\max}} \cdot \log_{B_{\max}})$ cases, which amounts to only a few hundred samples that can be completed within minutes.

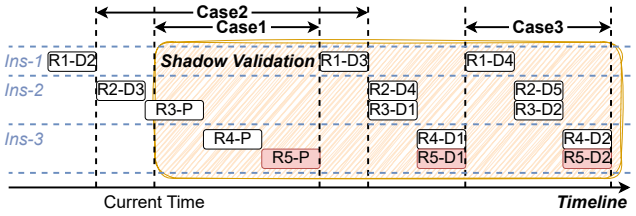



**Figure 6.** A shadow validation example with three cases.

### 5.3 Adding Request via Shadow Validation

Based on quantification, LLM-Mesh can estimate the time of each iteration under various loads, we next focus on how the SLO violation could occur and how to avoid it.

Specifically, as shown in Figure 6, when request-5 tries to join instance-3, there are three possible cases: (1) The prefill iteration of the new request (R5-P) is finished too late, making its own headroom negative. (2)Existing request (R1-D3) is delayed too late and becomes negative due to the prefill of new request. (3) After the new request, the target instance takes longer time to decode (R5-D1 and R5-D2), causing the aggregate time for a single decode iteration across all instances in the node to exceed TPOT SLO.

Therefore, when trying to add a new request to a target instance, LLM-Mesh performs a shadow validation to virtually add and simulate the future compute procedure. This is particularly important because LLM-Mesh prioritizes scheduling requests to compute-bound CPU instances. Considering the runtime fluctuations and the ever-growing sentence length during decode, LLM-Mesh overestimates each iteration by 10%. Finally, the instance will only accept the request if none of the above cases occur in the simulation. Otherwise, LLM-Mesh will retry the validation on other instances, including creating a new instance to serve the new request.

## 6 Hazard-Aware Memory Subsystem

### 6.1 Characterizing Memory Demands

The memory demand of one instance consists of model parameters and KV-cache, while the latter is dynamic and hard to determine since the final output length of each request is hard to know in advance. To avoid memory overprovisioning, LLM-Mesh adopts an optimistic estimation, assuming that each request's final output length is equal to the average output length $\bar{O}$ obtained from the historical logs. Additionally, to increase the robustness in cases with fewer requests, LLM-Mesh introduces a minimum total length hyper-parameter $L_{min}$, which is set to the maximum sentence length in practice.

Suppose there is a model instance where each token has a KV-cache size of $C$. If $R$ requests are currently running, with the $r$-th request having an input length of $I_r$ and having generated $O_r$ tokens, the memory requirement of KV-cache is:

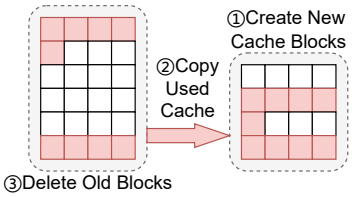



**Figure 7.** KV-cache scaling.

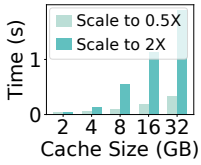



**Figure 8.** Scaling overhead on GPU.

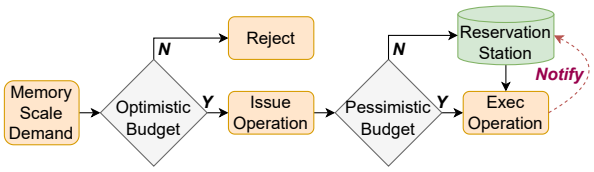



**Figure 9.** Flowchart of memory scaling operation.

$$M_{require} = C \cdot \max(\sum_{r=1}^{R}(I_r + \max(O_r, \bar{O})), L_{min}) \quad (2)$$

### 6.2 KV-Cache Scaling via Watermark

As the KV-cache demand fluctuates with each received or completed request, LLM-Mesh should dynamically respond by adjusting the allocated memory resource accordingly. However, we find that the adjusting procedure incurs non-negligible overhead. As demonstrated in Figure 7, based on the widely adopted paged-attention mechanism [29], scaling it requires creating new blocks, and copy the original KV-cache from old blocks to new blocks. As evaluated in Figure 8, scaling the original 32GB KV-cache space down to 16GB or up to 64GB requires 0.3 s and 1.9 s, respectively.

To reduce the scaling overhead, LLM-Mesh adopts an early scale-up and lazy scale-down strategy to avoid frequent adjustments. Specifically, it utilizes a watermark hyperparameter $w$, which is used to calculate the recommended value of KV-cache $M_{recommend} \leftarrow M_{require} \cdot (1 + w\%)$. Suppose the current KV-cache size is $M_{cur}$. When adding a new request and the current cache is insufficient ($M_{cur} < M_{require}$), LLM-Mesh scales up directly to $M_{recommend}$. This reserves space for subsequent incoming requests and prevents frequent scaling. When a request completes, LLM-Mesh defers scaling down the KV-cache unless the recommended value falls below the watermark ($M_{recommend} \cdot (1 + w\%) < M_{cur}$). This helps mitigate the ping-pong effect caused by load fluctuations.

### 6.3 Inter-Instance Scaling Orchestration

In a shared environment, each instance dynamically scales its KV-cache while also handling model loading and unloading. As a result, a node simultaneously undergoes multiple memory scaling operations, all of which are inherently asynchronous due to their execution latency. To efficiently manage memory adjustments and respond to fluctuations in real time, as illustrated in Figure 9, LLM-Mesh adopts an approach combining optimistic budgeting and pessimistic scheduling, inspired by Tomasulo's out-of-order execution [53], to maximize parallel execution of operations while avoiding hazards.

LLM-Mesh maintains an optimistic total memory budget within a node. When handling a scale-down demand, it preemptively lowers the budget and issues a corresponding operation. This budget adjustment is optimistic because the actual memory release only takes effect once the operation completes. Conversely, for a scale-up demand, it first checks whether the current budget can be increased to fit the needs. If it does, the budget is updated, and an operation is issued.

However, parallel execution introduces hazards, such as a scale-up immediately following a scale-down, which could lead to OOM errors. To avoid such risks, LLM-Mesh employs a pessimistic global memory tracking mechanism to determine whether to execute each issued operation. In this scheme, instances undergoing scale-down are accounted for based on their previous memory size. An issued scale-down operation will be directly executed. For a scale-up operation, if pessimistic tracking suggests a risk of OOM, the operation is placed in a reservation station rather than executing immediately. When a scale-down operation completes, it notifies the reservation station, which then re-evaluates the risk and attempts to execute any pending operations accordingly.

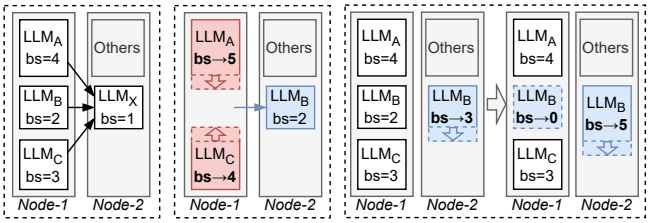
### 6.4 Intra-Instance Scaling Compromise

The orchestration mechanism may reject an instance's scale-up demand if there is insufficient available memory. To prevent the risk of rejection, when attempting to add a new request to a target instance, LLM-Mesh first performs a shadow check on whether the potential scale-up demand can be approved. To fully utilize all available memory, if the shadow check fails, LLM-Mesh will attempt to compromise the scale-up demand, allowing the request to be accepted as long as it can scale up to $M_{require}$ rather than $M_{recommend}$.

Additionally, although we have strengthened the robustness of the estimates for the KV-cache, there is still a possibility of underestimation. In this rare case, LLM-Mesh will attempt to scale up the cache again. If the attempt fails due to the node memory shortage, LLM-Mesh will evict and reschedule the request with the longest *headroom*.

## 7 Efficiency-Oriented Defragmentation

In a shared environment, the existence of neighbour instances may prevent one instance from scaling up to receive new request. Instead, the system scales out a new fragmented instance, as shown in Figure 10a, which leads to compute and memory efficiency degradation. LLM-Mesh helps reduce the fragmentation during scheduling with two strategies.




**(a)** Fragment **(b)** Proactive **(c)** Reactive (Two steps)

**Figure 10.** To avoid fragment cases, the incoming request A or C will trigger proactive manner. In reactive manner, subsequent B's requests are scheduled to Node-2 until bs=3, then instance in Node-1 is considered fragmented.

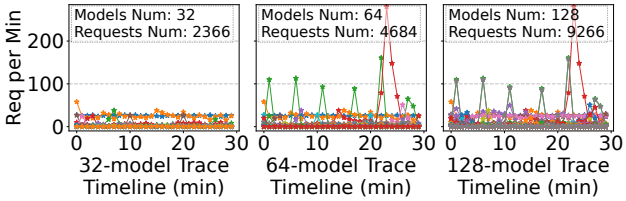



**Figure 11.** Azure Trace under different number of models.

### 7.1 Proactive Defragmentation with Preemption

When the scale-up is hindered by neighbour instances, LLM-Mesh allows one instance to preempt them to receive new request, as demonstrated in Figure 10b. The requests of the preempted instances will be rescheduled to other nodes.

However, the preemption can worsen fragmentation by disintegrating a previously enlarged neighbour instance. To prevent this, LLM-Mesh only allows an instance to preempt others with lower batch-size than itself and prioritizes the lowest one. Additionally, LLM-Mesh also runs a shadow validation based on the current headroom to ensure preempted requests can still meet their SLOs after rescheduling, allowing preemption only if the validation passes. As a result, even in a crowded environment, a small instance can still hold promise for growing into a larger one without affecting the existing large instances, thereby minimizing fragmentation.

### 7.2 Reactive Defragmentation with Bin-Packing

Although the fragmentation can be proactively minimized, an instance may still need to scale-out to create new fragmented instances. When there exist multiple instances of the same model, LLM-Mesh adopts a bin-packing scheduling strategy to eliminate the fragmented ones, prioritizing scheduling new requests to the instance with the highest batch-size. On one hand, high-batch instances have more opportunities to grow into larger instances through preemption, as discussed in §7.1, making them a more promising choice. On the other hand, low-batch instances are more likely to complete all current requests and free up resources, so avoiding scheduling new requests to smaller instances increases the likelihood of releasing their resources sooner.

Figure 10c illustrates this behavior: $LLM_B$, being the instance with minimal batch size requiring scaling, must provision a new instance on Node 2. When $LLM_B$ operates with $bs$=3 on Node 2 while $bs$=2 on Node 1, the latter instance is considered fragmented. Subsequent requests are then prioritized for scheduling to Node 2, enabling LLM-Mesh to reclaim Node 1's instance once its two remaining requests complete.

It is worth noting that existing work [21] typically uses a spread strategy to ensure load balancing and service quality. However, since LLM-Mesh accurately quantifies the performance of each instance, it can ensure satisfying SLOs while reducing fragmented instances through a bin-packing strategy.

## 8 Evaluation

### 8.1 Experimental Setup

**Testbed.** The testbed consists of 4 32-core Intel Xeon 6462C @3.3 GHz CPU nodes and 4 NVIDIA A100-80GB GPU nodes.

**Baselines.** We treat ServerlessLLM [21] as the first baseline, denoted as `sllm`, which only supports GPU nodes. Additionally, we have modified `sllm` to let it also utilize the CPU node. This is the second baseline, denoted as `sllm+cpu`.

**Datasets and Workloads.** The input and output length of each request are sampled from the Azure LLM Conversation dataset [43]. Since there is no public multi-LLM workload, following ServerlessLLM, we use Azure Serverless Trace [49], mapping each model to a function. We extracted the first 30-minute segment of the trace and uniformly sampled 32, 64, and 128 functions from it (illustrated in Figure 11).

**Models.** We use popular LLMs with 16-bit precision of different sizes: Llama-3.2-3B, Llama-2-7B, and Llama-2-13B.

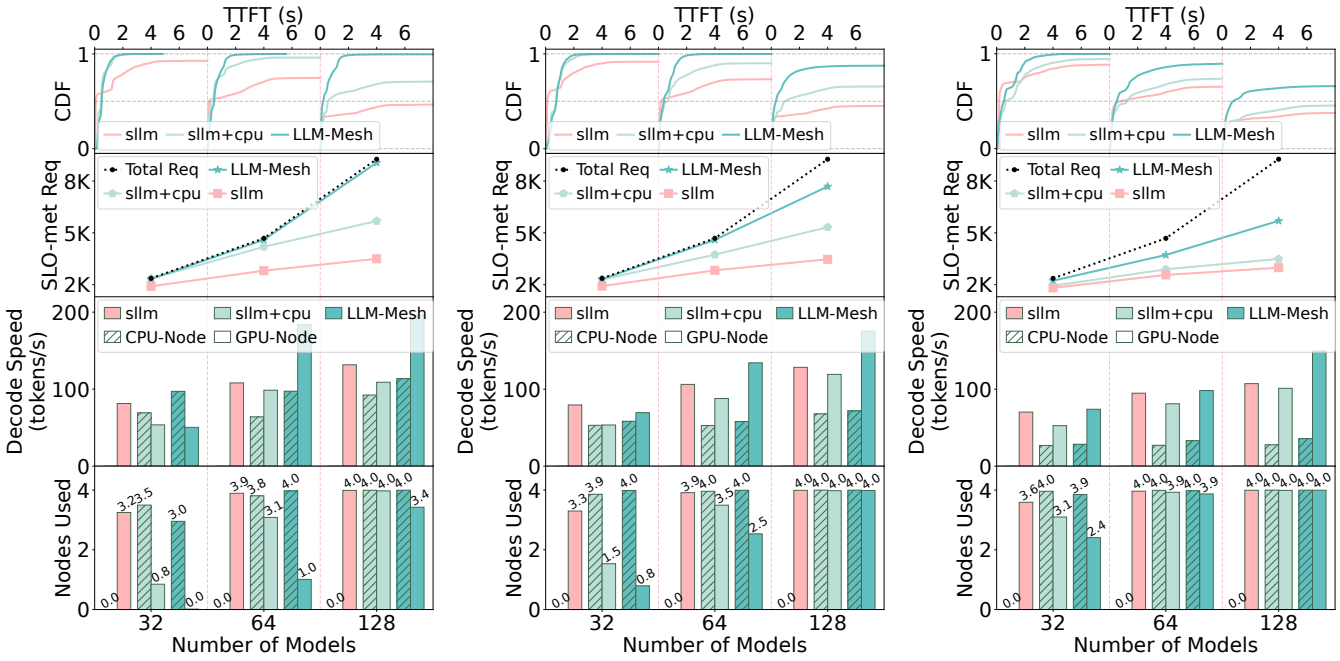
**SLOs.** For a request of input length $L$, as discussed in §2.1, we set TTFT and TPOT SLO to $\max(2, L/512)$ s and 0.25 s.

**Systems Behavior and Fairness.** All systems prioritize the CPU nodes. All models are cached in CPU memory, and the cold-start procedure is similar across all systems, as LLM-Mesh utilizes `sllm`'s loader to enable fast loading. The keep-alive threshold is set to 1 s and all systems use the same inference engines: vLLM 0.5.2 and OpenVINO 2024.6.0.

`sllm` triggers instance scale-out based on a default concurrency metric of 2, which leads to extreme inefficiency. Based on the profiling, we tried our best to conservatively tailor a set of higher concurrency values, which are (59, 15, 6) and (160, 32, 16) for 3B, 7B, and 13B models on CPU and GPU.

### 8.2 End-to-end Experiments

In this section, we present diverse performance metrics of LLM-Mesh under different model sizes and quantities, comparing it with `sllm` and `sllm+cpu`. Figure 12a shows the results for the 3B-sized cases, where 32, 64, and 128 replica models are generated from Llama-3.2-3B and mapped to




**(a)** 3B-sized cases. **(b)** 7B-sized cases. **(c)** 13B-sized cases.

**Figure 12.** Diverse performance metrics of each system under different model sizes and quantities.

the Azure Trace (Figure 11). Figure 12b and 12c depict the scenarios for the 7B-sized and the 13B-sized, respectively.

**LLM-Mesh uses less resource with higher per-node throughput.** When serving 32 3B-sized models in Figure 12a, LLM-Mesh consumes only 3.0 CPU nodes with 0 GPU, whereas `sllm` requires 3.2 GPU nodes. This is because `sllm` employs an exclusive approach, despite that GPUs offer high computational power. Using `sllm+cpu` can also reduce GPU usage by leveraging CPU resource, but it is less effective than LLM-Mesh. When serving 64 7B-sized models in Figure 12b, `sllm+cpu` can only reduce the GPU usage from 3.9 to 3.5, whereas LLM-Mesh can achieve a reduction to 2.5.

To further investigate the reasons behind LLM-Mesh's resource savings, we measured the average decode throughput per node. Compared to `sllm+cpu`, LLM-Mesh achieves throughput improvements by 6%-52% on CPU nodes and (-6)%-86% on GPU nodes. Note that the improvements can be negative since LLM-Mesh uses little GPU when serving 32 3B-sized models. The reasons for the improvements are twofold: First, LLM-Mesh can achieve and gain higher batch size. Second, `sllm+cpu` keeps the hardware compute resource idle during instance cold-start and recycle, while LLM-Mesh can schedule other instances instead.

Finally, as the number of models increases or model size grows, the resource usage gap between three systems gradually narrows. For example, when serving 128 13B-sized models (Figure 12c), each system exhausts all nodes. This is because, on one hand, the excessive load begins to saturate each system, and on the other hand, larger models diminish the sharing potential of LLM-Mesh (detailed in §8.4).

**LLM-Mesh achieves superior service capacity with shorter TTFT tail latency.** Under high loads, LLM-Mesh improves the number of SLO-compliant requests by 91% - 159% compared to `sllm` and by 44% - 63% compared to `sllm+cpu`. We also report the TTFT CDFs. Overall, LLM-Mesh does not experience worse tail latency due to resource sharing but does better. Unlike LLM-Mesh that uses shadow validation to verify the SLO, `sllm` and `sllm+cpu` experience long queuing under high load. Furthermore, when serving 128 models, the curves for `sllm` and `sllm+cpu` flatten at much lower percentiles, indicating that a significant portion of requests is dropped due to limited service capacity. Although LLM-Mesh also exhibits early flattening, it occurs much later.

### 8.3 Ablation Study

We further study the effectiveness of LLM-Mesh's design. Figure 13 shows the results when disabling each component while serving 64 7B-sized models. Disabling any component results in a significant increase in GPU resource usage. Notably, after disabling sharing, the SLO compliance rate drops substantially to 89%. This is because sharing is a key factor in increasing deployment density; without it, LLM-Mesh struggles to handle such a large number of models simultaneously.

From the truncated timeline of GPU usage, we observe that after disabling the CPU, GPU usage is consistently high,

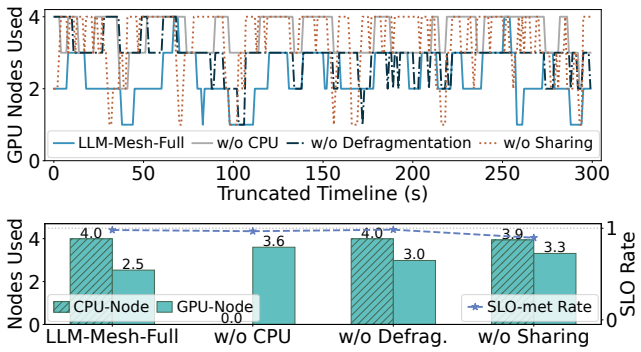



**Figure 13.** The resource usage and SLO compliance rate when disabling each component of LLM-Mesh.

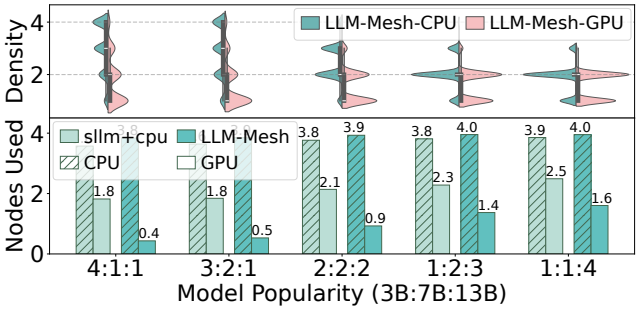



**Figure 14.** Performance when various sized models co-exist.

whereas LLM-Mesh-full rarely exhausts all four GPU nodes. Additionally, after disabling defragmentation, when handling fluctuating loads (at 50 s and 250 s) and after load spikes, the GPU usage is notably higher compared to LLM-Mesh-full. This is because it creates fragmented instances to handle the surge, which cannot be reclaimed promptly.

### 8.4 Mixed Deployment

Considering that real-world scenarios involve models of various sizes running simultaneously, we evaluate the effectiveness of LLM-Mesh in handling a mix of different model sizes in Figure 14. When serving a combination of 3B, 7B, and 13B models, LLM-Mesh consistently requires fewer GPU nodes compared to `sllm+cpu`. However, its efficiency is more sensitive to model popularity trends. When small models dominate (4:1:1) compared to when large models dominate (1:1:4), there is a 1.2 GPU node usage difference, whereas sllm+cpu exhibits a smaller gap of only 0.7. This is because larger models impose higher computational and memory demands, limiting the efficiency of LLM-Mesh's sharing mechanism. When the model ratio is 1:1:4, both each node can only deploy around two model instances concurrently, whereas under a 4:1:1 ratio, it can support three to four instances. Since most popular models are relatively small [3], LLM-Mesh can achieve significant reductions in resource usage in real-world deployments.

## 9 Related Work

**Heterogeneous serverless computing.** Designing serverless systems with heterogeneous hardware [44, 47] offers significant opportunities. Molecule [19] enables serverless computing to run seamlessly across heterogeneous computers, DSCS-Serverless [35] leverages programmable accelerators to unlock the potential of data centers. In the context of LLM serving, LLM-Mesh also identifies opportunities to leverage heterogeneous hardware effectively.

**Model serving systems.** Before the rise of LLM, traditional model serving systems [17, 18, 20, 30, 32]—such as Clockwork [22], Cocktail [23], and SHEPHERD [62]—had introduced numerous optimizations in scheduling and resource management. However, traditional models differ significantly from LLMs in their resource demands and execution patterns. The latter defines SLOs at token-level and executes in a multi-iteration manner, necessitating the specialized serving systems.

In response, a wave of LLM-oriented solutions [36, 40, 42, 56, 59] has emerged. vLLM [29] enhances memory efficiency with paged-attention, Llumnix [51] dynamically schedules requests across instances. These works primarily focus on high-load scenarios with a single LLM. Meanwhile, under serverless scenarios, ServerlessLLM [21] and Medusa [61] improve cold-start but still allocate dedicated GPUs for each LLM. LLM-Mesh focuses on resource management and is orthogonal to them.

**CPU-assisted LLM inference.** Given the scarcity of GPUs, many works [41, 57] explore leveraging CPUs for help. PowerInfer [50] and FastDecode [24] offload model parameters and KV-cache to the CPU, while CaraServe [31] uses CPU to mask cold-start overhead. They still rely on GPUs as the base and require CPUs and GPUs to be tight-coupled. In contrast, LLM-Mesh identifies CPU-independent serving potential and transparently utilizes heterogeneous resources through unified management.

## 10 Conclusion

We propose LLM-Mesh, an elastic-sharing serverless LLM inference scheme. Motivated by evolving hardware architectures and real-world workload characteristics, LLM-Mesh brings a new solution in the face of GPU scarcity. We consider LLM-Mesh as a first step in applying the serverless paradigm to explore transparent sharing of heterogeneous resource for LLM inference. As hardware continues to advance, more opportunities will emerge.

## References


[1] 2023. meta-llama/Llama-2-7b-hf · Hugging Face. https://huggingface.co/meta-llama/Llama-2-7b-hf.

[2] 2023. Phi-2: The surprising power of small language models - Microsoft Research. https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/.

[3] 2024. Open Source Ai Year In Review 2024 - a Hugging Face Space by huggingface. https://huggingface.co/spaces/huggingface/open-source-ai-year-in-review-2024?day=2.

[4] 2025. ChatGPT | OpenAI. https://openai.com/chatgpt/overview/.

[5] 2025. Deploy models as serverless APIs - Azure Machine Learning | Microsoft Learn. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-deploy-models-serverless.

[6] 2025. Host your LLMs on Cloud Run | Google Cloud Blog. https://cloud.google.com/blog/products/application-development/run-your-ai-inference-applications-on-cloud-run-with-nvidia-gpus.

[7] 2025. Inference API (serverless) - Hugging Face. https://huggingface.co/inference-api/serverless.

[8] 2025. Intel® Distribution of OpenVINO™ Toolkit. https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html.

[9] 2025. Intel® Xeon® 6 - 2 | Performance Index. https://edc.intel.com/content/www/us/en/products/performance/benchmarks/intel-xeon-6/.

[10] 2025. Meet Claude Anthropic. https://www.anthropic.com/claude.

[11] 2025. TensorRT-LLM. https://github.com/NVIDIA/TensorRT-LLM.

[12] 2025. What Is Intel® Advanced Matrix Extensions (Intel® AMX)? – Intel. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-amx.html.

[13] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 117–134. https://www.usenix.org/conference/osdi24/presentation/agrawal

[14] Ahsan Ali, Riccardo Pinciroli, Feng Yan, and Evgenia Smirni. 2020. Batch: machine learning inference serving on serverless platforms with adaptive batching. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, Christine Cuicchi, Irene Qualters, and William T. Kramer (Eds.). IEEE/ACM, 69. https://doi.org/10.1109/SC41405.2020.00073

[15] Cen Chen, Kenli Li, Aijia Ouyang, Zeng Zeng, and Keqin Li. 2018. GFlink: An In-Memory Computing Architecture on Heterogeneous CPU-GPU Clusters for Big Data. *IEEE Trans. Parallel Distributed Syst.* 29, 6 (2018), 1275–1288. https://doi.org/10.1109/TPDS.2018.2794343

[16] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. arXiv:2107.03374 [cs.LG] https://arxiv.org/abs/2107.03374

[17] Seungbeom Choi, Sunho Lee, Yeonjae Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. 2022. Serving Heterogeneous Machine Learning Models on Multi-GPU Servers with Spatio-Temporal Sharing. In *Proceedings of the 2022 USENIX Annual Technical Conference, USENIX ATC 2022, Carlsbad, CA, USA, July 11-13, 2022*, Jiri Schindler and Noa Zilberman (Eds.). USENIX Association, 199–216. https://www.usenix.org/conference/atc22/presentation/choi-seungbeom

[18] Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, Aditya Akella and Jon Howell (Eds.). USENIX Association, 613–627. https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw

[19] Dong Du, Qingyuan Liu, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2022. Serverless computing on heterogeneous computers. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch (Eds.). ACM, 797–813. https://doi.org/10.1145/3503222.3507732

[20] Jiarui Fang, Yang Yu, Chengduo Zhao, and Jie Zhou. 2021. TurboTransformers: an efficient GPU serving system for transformer models. In *PPoPP '21: 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Virtual Event, Republic of Korea, February 27- March 3, 2021*, Jaejin Lee and Erez Petrank (Eds.). ACM, 389–402. https://doi.org/10.1145/3437801.3441578

[21] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 135–153. https://www.usenix.org/conference/osdi24/presentation/fu

[22] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like Clockwork: Performance Predictability from the Bottom Up. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 443–462. https://www.usenix.org/conference/osdi20/presentation/gujarati

[23] Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Bikash Sharma, Mahmut Taylan Kandemir, and Chita R. Das. 2022. Cocktail: A Multidimensional Optimization for Model Serving in Cloud. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, Amar Phanishayee and Vyas Sekar (Eds.). USENIX Association, 1041–1057. https://www.usenix.org/conference/nsdi22/presentation/gunasekaran

[24] Jiaao He and Jidong Zhai. 2024. FastDecode: High-Throughput GPU-Efficient LLM Serving using Heterogeneous Pipelines. *CoRR* abs/2403.11421 (2024). https://doi.org/10.48550/ARXIV.2403.11421 arXiv:2403.11421

[25] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. 2019. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the 2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, Dahlia Malkhi and Dan Tsafrir (Eds.). USENIX Association, 947–960. https://www.usenix.org/conference/atc19/presentation/jeon

[26] Yimin Jiang, Yibo Zhu, Chang Lan, Bairen Yi, Yong Cui, and Chuanxiong Guo. 2020. A Unified Architecture for Accelerating Distributed DNN Training in Heterogeneous GPU/CPU Clusters. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 463–479.

https://www.usenix.org/conference/osdi20/presentation/jiang

[27] Artjom Joosen, Ahmed Hassan, Martin Asenov, Rajkarn Singh, Luke Nicholas Darlow, Jianfeng Wang, and Adam Barker. 2023. How Does It Function?: Characterizing Long-term Trends in Production Serverless Workloads. In *Proceedings of the 2023 ACM Symposium on Cloud Computing, SoCC 2023, Santa Cruz, CA, USA, 30 October 2023 - 1 November 2023*. ACM, 443–458. https://doi.org/10.1145/3620678.3624783

[28] Aditya K. Kamath, Ramya Prabhu, Jayashree Mohan, Simon Peter, Ramachandran Ramjee, and Ashish Panwar. 2024. POD-Attention: Unlocking Full Prefill-Decode Overlap for Faster LLM Inference. *CoRR* abs/2410.18038 (2024). https://doi.org/10.48550/ARXIV.2410.18038 arXiv:2410.18038

[29] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 611–626. https://doi.org/10.1145/3600006.3613165

[30] Yunseong Lee, Alberto Scolari, Byung-Gon Chun, Marco Domenico Santambrogio, Markus Weimer, and Matteo Interlandi. 2018. PRETZEL: Opening the Black Box of Machine Learning Prediction Serving Systems. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, Andrea C. Arpaci-Dusseau and Geoff Voelker (Eds.). USENIX Association, 611–626. https://www.usenix.org/conference/osdi18/presentation/lee

[31] Suyi Li, Hanfeng Lu, Tianyuan Wu, Minchen Yu, Qizhen Weng, Xusheng Chen, Yizhou Shan, Binhang Yuan, and Wei Wang. 2024. CaraServe: CPU-Assisted and Rank-Aware LoRA Serving for Generative LLM Inference. *CoRR* abs/2401.11240 (2024). https://doi.org/10.48550/ARXIV.2401.11240 arXiv:2401.11240

[32] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In *17th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2023, Boston, MA, USA, July 10-12, 2023*, Roxana Geambasu and Ed Nightingale (Eds.). USENIX Association, 663–679. https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

[33] 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 the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*, Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (Eds.). mlsys.org. https://proceedings.mlsys.org/paper_files/paper/2024/hash/42a452cbafa9dd64e9ba4aa95cc1ef21-Abstract-Conference.html

[34] Chiheng Lou, Sheng Qi, Chao Jin, Dapeng Nie, Haoran Yang, Xuanzhe Liu, and Xin Jin. 2025. Towards Swift Serverless LLM Cold Starts with ParaServe. arXiv:2502.15524 [cs.DC] https://arxiv.org/abs/2502.15524

[35] Rohan Mahapatra, Soroush Ghodrati, Byung Hoon Ahn, Sean Kinzer, Shu-Ting Wang, Hanyang Xu, Lavanya Karthikeyan, Hardik Sharma, Amir Yazdanbakhsh, Mohammad Alian, and Hadi Esmaeilzadeh. 2024. In-Storage Domain-Specific Acceleration for Serverless Computing. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 530–548. https://doi.org/10.1145/3620665.3640413

[36] Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. 2024. SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 932–949. https://doi.org/10.1145/3620666.3651335

[37] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2024. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 1112–1127. https://doi.org/10.1145/3620665.3640411

[38] Deepak Narayanan, Keshav Santhanam, Fiodar Kazhamiaka, Amar Phanishayee, and Matei Zaharia. 2020. Heterogeneity-Aware Cluster Scheduling Policies for Deep Learning Workloads. In *14th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2020, Virtual Event, November 4-6, 2020*. USENIX Association, 481–498. https://www.usenix.org/conference/osdi20/presentation/narayanan-deepak

[39] Nevine Nassif, Ashley O. Munch, Carleton L. Molnar, Gerald Pasdast, Sitaraman V. Lyer, Zibing Yang, Oscar Mendoza, Mark Huddart, Srikrishnan Venkataraman, Sireesha Kandula, Rafi Marom, Alexandra M. Kern, William J. Bowhill, David R. Mulvihill, Srikanth Nimmagadda, Varma Kalidindi, Jonathan Krause, Mohammad M. Haq, Roopali Sharma, and Kevin Duda. 2022. Sapphire Rapids: The Next-Generation Intel Xeon Scalable Processor. In *IEEE International Solid-State Circuits Conference, ISSCC 2022, San Francisco, CA, USA, February 20-26, 2022*. IEEE, 44–46. https://doi.org/10.1109/ISSCC42614.2022.9731107

[40] Hyungjun Oh, Kihong Kim, Jaemin Kim, Sungkyun Kim, Junyeol Lee, Du-Seong Chang, and Jiwon Seo. 2024. ExeGPT: Constraint-Aware Resource Scheduling for LLM Inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 369–384. https://doi.org/10.1145/3620665.3640383

[41] Daon Park and Bernhard Egger. 2024. Improving Throughput-oriented LLM Inference with CPU Computations. In *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques, PACT 2024, Long Beach, CA, USA, October 14-16, 2024*. ACM, 233–245. https://doi.org/10.1145/3656019.3676949

[42] 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, ASPLOS 2024, La Jolla, CA, USA, 27 April 2024- 1 May 2024*, Rajiv Gupta, Nael B. Abu-Ghazaleh, Madan Musuvathi, and Dan Tsafrir (Eds.). ACM, 207–222. https://doi.org/10.1145/3620666.3651329

[43] 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 *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 118–132. https://doi.org/10.1109/ISCA59077.2024.00019

[44] Tobias Pfandzelter, Aditya Dhakal, Eitan Frachtenberg, Sai Rahul Chalamalasetti, Darel Emmot, Ninad Hogade, Rolando Pablo Hong Enriquez, Gourav Rattihalli, David Bermbach, and Dejan S. Milojicic. 2023. Kernel-as-a-Service: A Serverless Programming Model for Heterogeneous Hardware Accelerators. In *Proceedings of the 24th International*

*Middleware Conference, Middleware 2023, Bologna, Italy, December 11-15, 2023*. ACM, 192–206. https://doi.org/10.1145/3590140.3629115

[45] Morgane Rivière, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar, Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozinska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucinska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju-yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjösund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, and Lilly McNealus. 2024. Gemma 2: Improving Open Language Models at a Practical Size. *CoRR* abs/2408.00118 (2024). https://doi.org/10.48550/ARXIV.2408.00118 arXiv:2408.00118

[46] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. INFaaS: Automated Model-less Inference Serving. In *Proceedings of the 2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 397–411. https://www.usenix.org/conference/atc21/presentation/romero

[47] Francisco Romero, Mark Zhao, Neeraja J. Yadwadkar, and Christos Kozyrakis. 2021. Llama: A Heterogeneous & Serverless Framework for Auto-Tuning Video Analytics Pipelines. In *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, Carlo Curino, Georgia Koutrika, and Ravi Netravali (Eds.). ACM, 1–17. https://doi.org/10.1145/3472883.3486972

[48] Rohan Basu Roy, Tirthak Patel, and Devesh Tiwari. 2022. IceBreaker: warming serverless functions better with heterogeneity. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch (Eds.). ACM, 753–767. https://doi.org/10.1145/3503222.3507750

[49] Mohammad Shahrad, Rodrigo Fonseca, Iñigo Goiri, Gohar Irfan Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. 2020. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *Proceedings of the 2020 USENIX Annual Technical Conference, USENIX ATC 2020, July 15-17, 2020*, Ada Gavrilovska and Erez Zadok (Eds.). USENIX Association, 205–218. https://www.usenix.org/conference/atc20/presentation/shahrad

[50] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024*, Emmett Witchel, Christopher J. Rossbach, Andrea C. Arpaci-Dusseau, and Kimberly Keeton (Eds.). ACM, 590–606. https://doi.org/10.1145/3694715.3695964

[51] 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 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 173–191. https://www.usenix.org/conference/osdi24/presentation/sun-biao

[52] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, Pouya Tafti, Léonard Hussenot, Pier Giuseppe Sessa, Aakanksha Chowdhery, Adam Roberts, Aditya Barua, Alex Botev, Alex Castro-Ros, Ambrose Slone, Amélie Héliou, Andrea Tacchetti, Anna Bulanova, Antonia Paterson, Beth Tsai, Bobak Shahriari, Charline Le Lan, Christopher A. Choquette-Choo, Clément Crepy, Daniel Cer, Daphne Ippolito, David Reid, Elena Buchatskaya, Eric Ni, Eric Noland, Geng Yan, George Tucker, George-Christian Muraru, Grigory Rozhdestvenskiy, Henryk Michalewski, Ian Tenney, Ivan Grishchenko, Jacob Austin, James Keeling, Jane Labanowski, Jean-Baptiste Lespiau, Jeff Stanway, Jenny Brennan, Jeremy Chen, Johan Ferret, Justin Chiu, Justin Mao-Jones, Katherine Lee, Kathy Yu, Katie Millican, Lars Lowe Sjoesund, Lisa Lee, Lucas Dixon, Machel Reid, Maciej Mikuła, Mateo Wirth, Michael Sharman, Nikolai Chinaev, Nithum Thain, Olivier Bachem, Oscar Chang, Oscar Wahltinez, Paige Bailey, Paul Michel, Petko Yotov, Rahma Chaabouni, Ramona Comanescu, Reena Jana, Rohan Anil, Ross McIlroy, Ruibo Liu, Ryan Mullins, Samuel L Smith, Sebastian Borgeaud, Sertan Girgin, Sholto Douglas, Shree Pandya, Siamak Shakeri, Soham De, Ted Klimenko, Tom Hennigan, Vlad Feinberg, Wojciech Stokowiec, Yu hui Chen, Zafarali Ahmed, Zhitao Gong, Tris Warkentin, Ludovic Peran, Minh Giang, Clément Farabet, Oriol Vinyals, Jeff Dean, Koray Kavukcuoglu, Demis Hassabis, Zoubin Ghahramani, Douglas Eck, Joelle Barral, Fernando Pereira, Eli Collins, Armand Joulin, Noah Fiedel, Evan Senter, Alek Andreev, and Kathleen Kenealy. 2024. Gemma: Open Models Based on Gemini Research and Technology. arXiv:2403.08295 [cs.CL] https://arxiv.org/abs/2403.08295

[53] R. M. Tomasulo. 1967. An Efficient Algorithm for Exploiting Multiple Arithmetic Units. *IBM Journal of Research and Development* 11, 1 (1967), 25–33. https://doi.org/10.1147/rd.111.0025

[54] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288 [cs.CL] https://arxiv.org/abs/2307.09288

[55] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, Amar Phanishayee and Vyas Sekar (Eds.). USENIX Association, 945–960. https://www.usenix.org/conference/nsdi22/presentation/weng

[56] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings*

*of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024*, Emmett Witchel, Christopher J. Rossbach, Andrea C. Arpaci-Dusseau, and Kimberly Keeton (Eds.). ACM, 640–654. https://doi.org/10.1145/3694715.3695948

[57] Yi Xu, Ziming Mao, Xiangxi Mo, Shu Liu, and Ion Stoica. 2024. Pie: Pooling CPU Memory for LLM Inference. *CoRR* abs/2411.09317 (2024). https://doi.org/10.48550/ARXIV.2411.09317 arXiv:2411.09317

[58] Yanan Yang, Laiping Zhao, Yiming Li, Huanyu Zhang, Jie Li, Mingyang Zhao, Xingzhen Chen, and Keqiu Li. 2022. INFless: a native serverless system for low-latency, high-throughput inference. In *ASPLOS '22: 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, 28 February 2022 - 4 March 2022*, Babak Falsafi, Michael Ferdman, Shan Lu, and Thomas F. Wenisch (Eds.). ACM, 768–781. https://doi.org/10.1145/3503222.3507709

[59] 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 2022, Carlsbad, CA, USA, July 11-13, 2022*, Marcos K. Aguilera and Hakim Weatherspoon (Eds.). USENIX Association, 521–538. https://www.usenix.org/conference/osdi22/presentation/yu

[60] Minchen Yu, Rui Yang, Chaobo Jia, Zhaoyuan Su, Sheng Yao, Tingfeng Lan, Yuchen Yang, Yue Cheng, Wei Wang, Ao Wang, and Ruichuan Chen. 2025. $\lambda$Scale: Enabling Fast Scaling for Serverless Large Language Model Inference. arXiv:2502.09922 [cs.DC] https://arxiv.org/abs/2502.09922

[61] Shaoxun Zeng, Minhui Xie, Shiwei Gao, Youmin Chen, and Youyou Lu. 2025. Medusa: Accelerating Serverless LLM Inference with Materialization. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 653–668. https://doi.org/10.1145/3669940.3707285

[62] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. 2023. SHEPHERD: Serving DNNs in the Wild. In *20th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2023, Boston, MA, April 17-19, 2023*, Mahesh Balakrishnan and Manya Ghobadi (Eds.). USENIX Association, 787–808. https://www.usenix.org/conference/nsdi23/presentation/zhang-hong

[63] Laiping Zhao, Yanan Yang, Yiming Li, Xian Zhou, and Keqiu Li. 2021. Understanding, predicting and scheduling serverless workloads under partial interference. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin (Eds.). ACM, 22. https://doi.org/10.1145/3458817.3476215

[64] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. 2024. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=BOfDKxfwt0

[65] 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 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin