AcceLLM: Accelerating LLM Inference using Redundancy for Load Balancing and Data Locality

Ilias Bournias [1] [*]   **Lukas Cavigelli** [1]   **Georgios Zacharopoulos** [1] [*]

ABSTRACT

Large Language Model (LLM) inference on large-scale systems is expected to dominate future cloud infrastructures. Efficient LLM inference in cloud environments with numerous AI accelerators is challenging, necessitating extensive optimizations for optimal performance. Current systems batch prefill and decoding to boost throughput but encounter latency issues, while others disaggregate these phases, leading to resource underutilization. We propose AcceLLM, a novel method addressing latency and load balancing, inspired by the cache data management. It strategically utilizes redundant data to enhance inference via load balancing and optimal hardware use. Simulated evaluations on Nvidia H100 GPU and Huawei Ascend 910B2 show AcceLLM surpasses state-of-the-art systems up to 30% in latency and efficiency, handling diverse workloads effectively.

1 INTRODUCTION

The surge in demand for Large Language Model (LLM) inference is driven by the widespread adoption of Natural Language Processing (NLP) applications across industries, including chatbots, virtual assistants, analytics, and automated content creation. This growth (Schillaci, 2024) is expected to further strain the hardware resources of cloud infrastructures.

Leading tech companies such as OpenAI, Microsoft, and Google utilize extensive GPU and TPU resources to handle LLM inference tasks (Patel & Ahmad, 2024), which involve two primary computational stages: prefill and decoding. Prefill processes the initial prompt, while decoding generates tokens sequentially, employing multi-head self-attention and feed-forward networks.

In a heterogeneous computing ecosystem, exemplified by a cloud infrastructure, high-speed interconnects enable efficient collaboration among diverse processors (McColl, 2017). Software mechanisms facilitate workload distribution and coordination across different levels, from hardware acceleration to task scheduling and load balancing.

However, current LLM inference systems experience latency spikes between output token generation (FasterTransformer; Kwon et al., 2023; Agrawal et al., 2023; Li et al., 2024; Zhao et al., 2024; Miao et al., 2023; Sheng et al., 2023; Strati et al., 2024a), adversely affecting user experience and necessitating over-provisioning of resources. To address this, researchers have explored separating prefill and decoding stages across different machines, or *instances* that

---
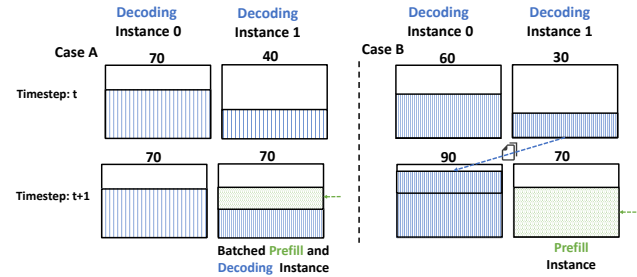[*]Equal contribution [1]Computing Systems Lab, Huawei, Zurich Research Center.

Figure 1. Peak Latency Challenge. Case A: Batched Prefill and Decoding resulting in a latency peak. Case B: Decoding data transfer overhead between instances due to a large prefill request.

can employ multiple devices (Hu et al., 2024; Patel et al., 2023; Zhong et al., 2024) and involve transferring a large state, often referred to as KV cache.

This approach aims to manage diverse task granularities and scale resource pools according to computational demands, mitigating latency peaks caused by lengthy prefill tasks interleaved with token generation. It also puts away with latency peaks that otherwise occur when the commonly much longer prefill tasks are executed between single token decoding/generation tasks. We visualize this challenge in Figure 1 (Case A), where an incoming prefill request necessitates batching and simultaneous computation alongside a decoding task on the same computing instance, resulting in increased latency.

However, the methods mentioned above *do not* tackle two main challenges that can significantly hinder performance: a) prefill-related latency peaks and b) lack of load balancing. The first is due to the inherent need to copy the LLM query's state (KV cache) before the decoding nodes can start producing output tokens, which in turn adds respective communication cost, as seen in Figure 1 (Case B). Furthermore, they fail to tackle challenges associated with load imbalances concerning both prefill and decoding requests across
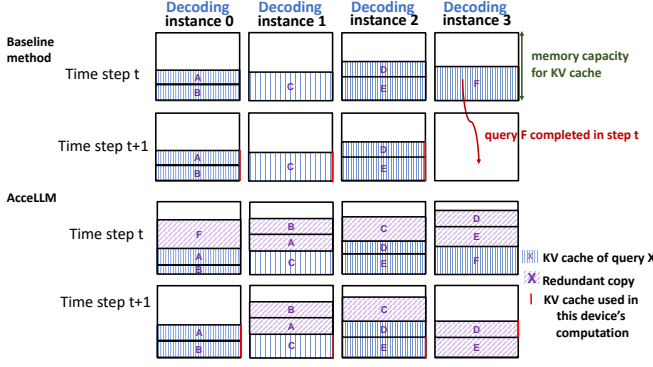
Figure 2. The KV cache of different queries is allocated on different instances. Query F (KV cache on Instance 3), is completed in time step t. In step $t + 1$ (and many later), device 3 is idle. We use the free space on the devices to keep redundant copies of the KV cache. After query F is completed, there is enough KV cache on every device to keep good **load balancing**.

devices, leading to underutilization of available hardware resources, as seen in Figure 2.

To overcome these challenges and enhance hardware resource utilization, we propose AcceLLM, a novel method offering data locality, load balancing, and dynamic instances by employing redundant data copies across computing instances. We provide the following contributions:

1. We propose a novel method —AcceLLM— leveraging **redundant data** in the form of KV cache copies across instances of multiple devices, drawing inspiration from the data management in caches and cloned computing.

2. We show how AcceLLM reduces latency and provides superior cost efficiency by achieving effective **load balancing** among nodes.

3. We show an **improved utilization** as instances can serve both, generation or prefill tasks, dynamically according to the workload demand, without requiring any internal adjustments of the LLM's weights.

4. We offer an **extensive evaluation** of AcceLLM on two leading simulated platforms, Nvidia's H100 GPU and Huawei's Ascend 910B2, and compare it against state-of-the-art methods.

2 RELATED WORK

The exceptionally high demand for serving LLM inference requests has lead to an increasing number of LLM serving systems. Conventional request-level batching systems, such as Faster-Transformer (FasterTransformer), employ a decode-prioritizing scheduling approach. In these systems, a batch of requests is processed by first completing the prefill phase for all requests, followed by scheduling their decode phase. The batch is considered complete only after all requests within it have finished decoding. While

this method optimizes the time between tokens, it significantly increases the queuing time for incoming requests, as they must wait for the current batch to complete before execution. Additionally, it reduces throughput since batch execution continues until the last request is finished, even if some requests complete earlier.

Orca (Yu et al., 2022) tackles this challenge by employing iteration-level scheduling. They optimize GPU resource utilization by batching the prefill and decoding phases of different requests in different phases. This approach minimizes redundant usage of High Bandwidth Memory (HBM). Similarly, Sarathi (Agrawal et al., 2023) and DeepSpeed (Holmes et al., 2024) adopt analogous strategies, but with slight variations. Sarathi chunks the prefill phase to align with the decoding phase, aiming to maximize GPU resource utilization and make it compute bound. This approach ensures uniform compute iterations, effectively minimizing pipeline bubbles. DeepSpeed further refines this by decomposing long prompts into smaller segments and scheduling them across multiple forward passes (iterations), with only the final pass performing generation.

Sarathi-Serve (Agrawal et al., 2024) utilizes chunked-prefills from Sarathi (Agrawal et al., 2023) for uninterrupted scheduling, enabling new batch additions without stalling ongoing decodes. This stall-free scheduling supports larger batch sizes and improves throughput while controlling latency. Vanilla LLM (vLLM) (Kwon et al., 2023) optimizes GPU memory by segmenting the KV cache by sequence length but prioritizes prefill, which increases decode latency. Infinite-LLM (Lin et al., 2024) employs DistAttention, a distributed attention algorithm, and proposes DistKV-LLM to manage KV caches across GPU and CPU memories in the data center. However, token generation is slowed as single requests may be partially served by multiple instances.

LoongServer (Wu et al., 2024) introduces elastic sequence parallelism by employing elastic scaling mechanisms and a scalable scheduling algorithm, dynamically adjusting resource allocation based on workload demands. It focuses on efficiently handling very long input sequences by flexibly managing the KV cache across various phases. However, like Infinite-LLM, it faces challenges related to data locality. Despite their distinct optimizations, these approaches batch the prefill and decode phases together, leading to reduced throughput and latency spikes during token generation.

To address the challenges of batching the prefill and decode phases, recent methods propose using separate instances for each phase. Splitwise (Patel et al., 2023) optimizes token generation by transferring KV caches from prefill to decode instances via high-speed GPU interconnects, employing a two-level scheduling system: a cluster-level scheduler assigns requests to machines, and a machine-level scheduler manages batching within each machine. Similarly, TetriInfer

(Hu et al., 2024) segments prompts into fixed-size chunks to maintain high compute efficiency and uses a two-level scheduling algorithm to reduce decode-phase bottlenecks. Additionally, it employs an LLM-based classifier, or "predict model," to categorize token lengths, enhancing resource allocation efficiency for each request.

DistServe (Zhong et al., 2024) allocates prefill and decoding tasks to separate GPUs, optimizing resource allocation and parallelism based on time-to-first-token and time-per-token requirements while exploring varied batching strategies. It periodically reallocates resources according to workload patterns, monitored by a profiler that tracks input/output lengths and arrival times.

DejaVu (Strati et al., 2024b) enables efficient KV cache streaming, disaggregating prefill and decode phases through micro-batching and KV cache swapping between GPU and CPU, reducing GPU memory demands and supporting larger batch sizes on limited hardware. To ensure fault tolerance, it duplicates KV caches in persistent or remote CPU memory, allowing restoration and resumption of inference after failures. Most methods dedicate instances to either prefill or decode tasks, without load balancing between them, which limits request throughput.

3 GENERATIVE LLM INFERENCE

Generative Large Language Model (LLM) inference has two main stages: prefill and decoding. In prefill, the model processes the user's prompt to generate the first token. Then, in decoding, the model iteratively generates tokens, each feeding into the next, until it hits the maximum token limit, a user-defined stop criterion, or produces an End Of Sentence (EOS) token, signaling the end of generation.

3.1 The Transformer Architecture

The Transformer block is the fundamental unit in LLM architectures. It includes the QKV Linear layer, which projects input data into queries (Q), keys (K), and values (V)(Chang et al., 2024). These projections feed into the multi-head self-attention mechanism, the main module in the Transformer block, allowing the model to focus on different parts of the input sequence with varying importance. Multi-head attention enables the model to process information from multiple subspaces concurrently. Following this is the Feed-Forward Neural Network (FFN), typically a three-layer network with an activation function in between. This general structure is consistent across Transformer implementations.

3.2 Prefill

Prefill constitutes a computationally demanding task (Patel et al., 2023), constrained by the processing capacity of GPU devices. Figure 3 illustrates both, the completion time of the
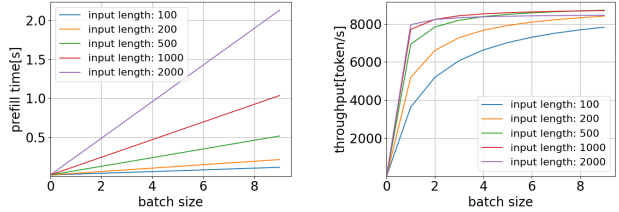

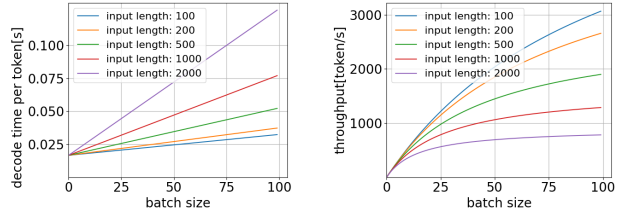*Figure 3.* Prefill-phase execution time and throughput.

Figure 4. Decoding-phase execution time and throughput.

prefill process showcasing its dependency on both prompt length and batching, and the number of generated tokens, also contingent on prompt length and batching. The throughput increases while batching more requests, until a certain plateau is reached as the compute operations outweigh the data transfers. To further increase throughput, parallelization must be employed, assigning batches of requests to different instances.

3.3 Decoding

Contrary to prefill, decoding is bounded by the HBM memory bandwidth as the amount of computation is minimal compared to the data transfer times from the memory to the compute units (Hong et al., 2024). In order to make better allocation of the GPU resources it is essential to apply batching between the decoding requests since it allows reuse of the LLM's weights for multiple requests. In Figure 4, we show the computation time and the token generation throughput for batched decoding.

As batch sizes increase, throughput rises due to fewer memory reads (Yu et al., 2022). Initially, this is independent of input length and with larger batch sizes, throughput stabilizes at distinct values for each input length. For longer inputs, the primary bottleneck becomes waiting for the loading of KV cache, as weight loading time that is independent of the input length becomes irrelevant due to the batching. In this important scenario, the compute time is largely proportional to the number of tokens.

Notably, this throughput limitation by the HBM bandwidth loading of the KV cache during decoding is also the reason why moving the KV cache from one device to another leads to a latency peak—simply as inter-device interconnect speeds are usually an order of magnitude lower than the local HBM bandwidth.

3.4 Target Metrics

We optimize four key metrics for serving LLM inference. The first, *Time to First Token (TTFT)*, measures the delay in generating the first token after receiving a prompt, critical for applications needing fast initial responses. The second, *Time Between Tokens (TBT)*, tracks token generation throughput, essential for responsive streaming. *Job Completion Time (JCT)* gauges the total time from prompt receipt to End-of-Sequence (EOS), assessing overall inference performance. Finally, *cost efficiency* is defined as throughput per instance, assuming cost correlates with runtime and instance count. While latency metrics (TTFT, TBT, JCT) reflect user experience, *cost efficiency* indicates user expense.

3.5 Challenges

3.5.1 Interference of Prefill and Decoding

Most recent state-of-the-art inference systems (Agrawal et al., 2023; Yu et al., 2022; Holmes et al., 2024) are batching prefill and decoding requests to enhance token generation throughput and fully utilize GPU computation capabilities. While this strategy maintains a relatively consistent TTFT compared to solely processing prefill requests, it results in a notable increase in TBT, as depicted in Figure 5.

This elevation in TBT occurs due to latency spikes caused by the arrival of one or more new requests which are in the prefill phase while an instance is engaged in decoding tasks. The spikes disrupt the smooth processing flow, causing delays in token generation for ongoing tasks. Consequently, this can lead to an increased JCT for each request.

3.5.2 Load Imbalance

The size of prompts and the quantity of generated tokens during the decoding phase are not predetermined for each request. Moreover, they vary significantly between requests. Consequently, there are considerable differences in the number of requests (batch size) handled by individual instances since some requests may complete earlier while others might take much longer time due to their decode phase.

The wide variance in request sizes poses a challenge when deploying multiple instances, potentially resulting in an uneven distribution of workload. This discrepancy arises because some instances, constrained by HBM bandwidth, may remain underutilized, leading to wasted hardware resources. Conversely, other instances may become overutilized, failing to leverage available resources effectively. These disparities, largely stemming from variations in batch sizes among instances, contribute to significant fluctuations in TBT (Figure 5) and ultimately result in an increased JCT. Addressing this imbalance is crucial for optimizing system performance and ensuring efficient resource utilization across all instances.
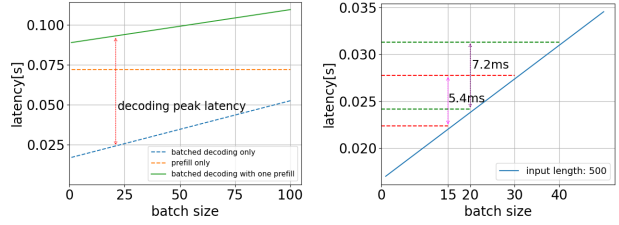

Figure 5. Left: Integrating batching with prefill in the decoding phase increases token generation latency by over 300%. Right: Imbalance arises when batching 40 requests per instance, increasing token generation by 7.2ms compared to parallel execution of the same requests across two instances with a batch size of 20.
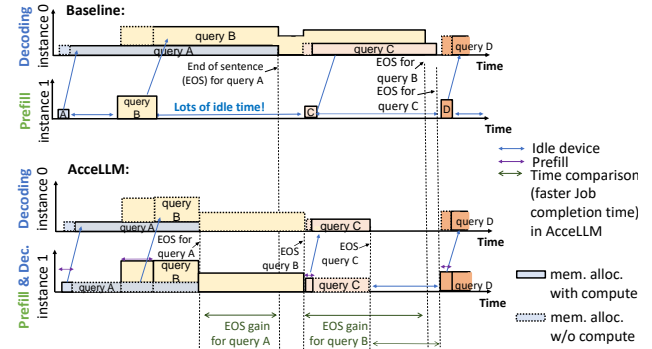

Figure 6. The new requests arrive at the same time in both baseline and AcceLLM. Baseline: the prefill device is idling a lot. AcceLLM: no device is idle at any time leading to better load balancing and as a result to strictly better (never worse) performance.

3.5.3 Dynamic Workload Handling

The varying rate at which new requests arrive makes it extremely challenging to allocate the available resources optimally in advance (Liu et al., 2024; Griggs et al., 2024; Hu et al., 2024). Certain state-of-the-art systems (Liu et al., 2024; Griggs et al., 2024; Srivatsa et al., 2024), based on the incoming prompt and a prediction model, decide about the amount of hardware resources that should be allocated. In some other approaches, where disaggregation is applied, the number of instances responsible for computing the prefill and the decode phase is predetermined (Zhong et al., 2024; Hu et al., 2024; Strati et al., 2024b; Patel et al., 2023) and it solely depends on the incoming request rate. In some of these approaches, (Patel et al., 2023; Hu et al., 2024) there are some extra static instances which can be used and serve either requests that are in the prefill phase or requests which are in the token generation phase depending on the incoming workload.

However, to the best of our knowledge, **none of the previous state-of-the-art works have instances which can be redirected from one phase to the other** (Strati et al., 2024b; Patel et al., 2023; Zhong et al., 2024; Hu et al., 2024), depending on the workload and the *currently* served requests in the decode phase. For redirecting one instance, first serving all the requests stored inside, or moving the KV caches stored in it to other instances is required (Srivatsa

4

et al., 2024). These approaches result in instances being idle for long periods of time (Figure 6) when no new requests arrive, or when the number of requests in the decode phase is very low, leading to the increase of TTFT, TBT and JCT. Furthermore, the migration of whole KV caches during decoding for the conversion, also leads to large latency peaks during requests serving making this approach impractical.

3.6 LLM Inference Models

Three computation models for LLM inference are investigated within the scope of this paper, that are specifically tailored for large-scale systems. A) Disaggregation model. Models that disaggregate the prefill and the decoding stages (Hu et al., 2024; Patel et al., 2023; Zhong et al., 2024) by employing separate instances dedicated to each stage. B) Load balancing model. Models that target equivalent distribution of workloads across all available resources, so that both better utilization and decreased latency can be achieved, as also demonstrated in Section 3.5.2. C) Redundancy model. Models that generate and hold redundant data that can be exploited during both the prefill and the decoding stages of LLM inference. We propose **AcceLLM**, a model that aims at achieving data locality and load balancing by using redundancy strategically in order to eliminate unnecessary communication among instances.

3.6.1 Disaggregation Model

In order to address the peak latency problem presented in Figure 1 (Section 1), models that disaggregate the prefill and decoding stages have been suggested (Hu et al., 2024; Patel et al., 2023; Zhong et al., 2024). This approach successfully tackles the latency resulting from a batched prefill and decoding instance, as seen in Case A (Figure 1), however it does not address Case B where there can be latency due to data transfer overhead among instances.

3.6.2 Load Balancing Model

In order to address load imbalance challenges, as mentioned in the previous section, a balanced workload distribution is suggested by our method. Moreover, as seen in Figure 5, a large batch size handled by a single instance, for instance batch size 40, would result into a larger latency equal to 7.2ms compared to two instances of batch size equal to 20 for any input length. Therefore, a balanced distribution of allocated resources can lead to faster inference times.

3.6.3 Redundancy Model

In order to address all the challenges mentioned above, we introduce a novel concept of performing LLM inference by generating redundant data across instances, inspired from the data management in caches. This model and our method is explained in detail in the following section.
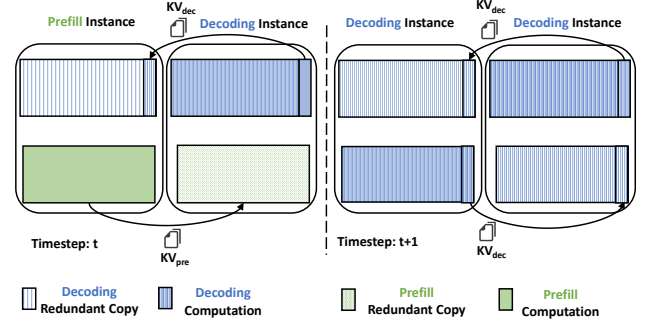

Figure 7. At timestep t, a new prefill is computed, and its newly created KV cache is transferred to the second instance. Simultaneously, decoding occurs in the second instance, and the newly computed KV cache lines are transferred to the first instance to ensure their updates. At timestep $t + 1$, when there is no prefill, both instances focus solely on decoding tasks, continuously synchronizing and updating the KV caches between each other.

4 ACCELLM

4.1 Methodology

Drawing upon the insights outlined in Section 3, we introduce AcceLLM, an LLM inference system leveraging redundant KV cache data to split prefill and decoding requests. This approach aims at minimizing latency through load balancing and the utilization of flexible instances capable of serving either for generation or prefill tasks, depending on the workload.

4.1.1 Dynamic Instances

In order to tackle the peak latency problem, the prefill and decode phase is disaggregated among instances (Patel et al., 2023; Strati et al., 2024b; Zhong et al., 2024; Hu et al., 2024). AcceLLM employs a similar strategy, however, contrary to previous state-of-the-art methods, where a prefill instance cannot fulfill decoding requests, and vice versa, AcceLLM supports dynamic instances capable of serving both phases, avoiding the peak latency issues by not serving them at the same time. By ensuring that no instances remain idle during computations, this approach achieves faster request latency (TTFT, TBT, JCT). Depending on the workload volume and the number of requests processed during the decode phase, instances can dynamically switch between decoding and prefilling roles, thereby optimizing system performance.

The scheduling manager within AcceLLM disaggregates the prefill and decode phase and dynamically assesses, depending on the incoming workload rate, whether an instance will compute prefill or decoding requests. This capability enables the system to better accommodate varying rates of incoming requests. Nevertheless, transitioning an instance from one phase of computation to the other is challenging, due to the large amount of complete KV cache transfers.

5

4.1.2 Redundant KV Caches

To overcome the hurdle of frequently moving complete KV caches between instances and to streamline the conversion of instances into prefill or decoding roles, we introduce the redundant KV caches method within AcceLLM, as seen in Figure 7.

During the prefill phase in AcceLLM, when a request is served from an instance, the entire generated KV cache is transferred to the instance designated for token generation (Zhong et al., 2024; Hu et al., 2024). Unlike previous state-of-the-art methods, AcceLLM retains a copy of the KV cache in the instance where prefilling occurred, provided there is adequate free space in the instance's memory. Additionally, the instance responsible for the decoding phase, during its processing sends newly computed KV cache lines back to the instance storing the KV cache copy where initially the prefill was computed, ensuring its continuous updating.

Once there are no prefill requests to be served in the instance originally designated for prefilling, it can be repurposed for decoding tasks. The updated KV caches stored there are then utilized for token generation, distributing the decoding workload across multiple instances to expedite token generation and avoiding this way keeping a few of them idle. This way, there is no need for full KV caches copies before transitioning a prefill to a decoding instance, which can result into high peak latency.

However, the newly created KV caches in the former prefill instance must be transferred back to the instance where decoding initially commenced. This is necessary to ensure that an instance remains available for prefilling when new requests arrive, without stalling requests or waiting for all the requests in the decode phase to complete. Furthermore, it ensures that requests in the decode phase are not delayed, as their processing can continue in the instance where decoding initially began.

When the system consists of two instances, redundancy of KV caches for nearly all served requests can be attained. This is because, for almost every request, after its prefill is served in one instance and its decoding in another, an updated KV cache copy is retained in both instances. The only scenario where a redundant copy cannot be maintained is when the KV cache has expanded to the extent that storage in a single decoding instance is insufficient, necessitating the involvement of additional instances for service.

In systems with more than two instances, half of the instances are allocated to either prefill or decoding tasks based on workload demands. As request rates rise and KV cache memory demands grow, some instances may replace redundant KV copies with new requests, causing a sequential shift into decoding roles. For example, in a four-instance setup
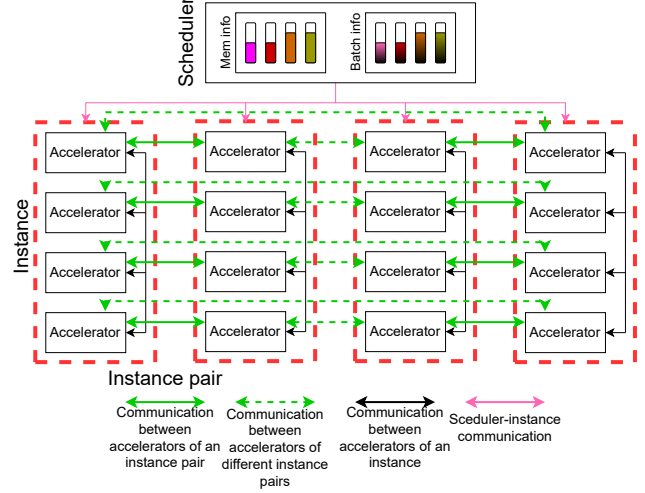

Figure 8. AcceLLM system: The instances are organized in pairs. All the instances are capable of serving prefill or decoding requests, but not both at the same time. The scheduler assigns the computation of the new prefill in the appropriate instance based on the free memory space. Batch size info is used to balance workload in the decoding phase.

lacking sufficient memory for redundancy, one instance may alternate between prefill and decode tasks, while the other three focus solely on decoding.

4.1.3 Load Balancing

Batching requests during token generation in the decoding phase is essential for maximizing accelerator resource usage, reducing the memory bandwidth pressure during these otherwise bandwidth-limited tasks (Recasens et al., 2024). This results in the preservation of a low TBT (Zhong et al., 2024). However, excessive batching of requests leads to an increase in the TBT. As also emphasized in (Hu et al., 2024; Recasens et al., 2024), it is equally important to balance the number of requests per batch per instance as much as possible, especially when multiple instances are available. With the EOS token appearing at varying iteration numbers for each request, load balancing between instances becomes even more unpredictable.

Additionally, it is crucial to ensure that the total length of requests within each batch is approximately equal. This is because, during the decoding phase, the only computation that varies between requests is the attention mechanism (Dao et al., 2022; Dao, 2023; Hong et al., 2024), which depends on the length of the KV cache.

With the assistance of redundant KV cache copies, AcceLLM achieves an effective balancing of the number of requests per batch among instances and evenly distributes requests according to their lengths. When a pair of instances function as decoding instances, AcceLLM ensures that the number of requests batched in each instance is approxi-

mately equal (assuming sufficient memory space). This approach leads to a reduction in TBT without necessitating additional movements of complete KV caches, while maximizing the utilization of available hardware resources. To balance the workload between pairs of instances, additional KV cache copies are generated gradually over time, provided that the communication bandwidth between instances isn't already saturated and there's no increase in latency overhead during token generation.

4.2 Technical Implementation

The high-level overview of AcceLLM is depicted in Figure 8. The system consists of instances grouped in pairs, capable of handling either prefill or decoding requests depending on the workload, but not both simultaneously. Every instance includes a full LLM model replication.

4.2.1 Instance Pair

As shown in Figure 8, the instances in AcceLLM are grouped in pairs for these experiments. With sufficient memory in each instance, nearly all request KV caches are replicated across the two paired instances. This setup enables one instance to handle prefills for new requests while the other continues decoding tokens for existing requests. During periods with no new requests, both instances switch to decoding, ensuring no idle time and allowing smooth transitions between operations with minimal latency overhead. When both instances are decoding, near-perfect load balancing is achieved, as redundant KV caches allow flexible batch size adjustments without latency penalties, even if imbalances arise due to completed requests.

Table 1. Accelerator Device Specifications

Device	fp16-TFLOPS	HBM cap.	HBM BW	local conn.
910B2	400	64 GB	1.8 TB/s	392 GB/s
H100	989	80 GB	3.35 TB/s	900 GB/s

Table 2. Workload Characteristics

Workload	Prefill	Decoding	Mean Value
Light	20-500	20-500	250
Mixed	20-1000	20-1000	500
Heavy	500-1000	500-1000	750

4.2.2 Scheduling Manager

The scheduling manager in AcceLLM directs requests and handles instance conversions. When a new request arrives, it routes it to one instance in a pair, switching it to prefill computation while flagging the second instance to continue token generation for all stored requests including the re-
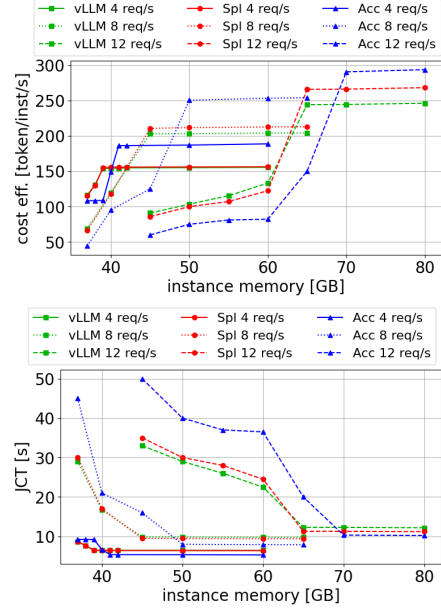

Figure 9. Memory requirements for mixed workload for a cluster consisting of four instances.
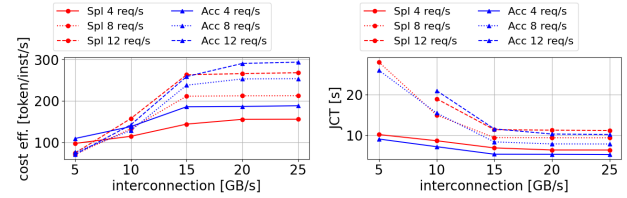

Figure 10. Interconnection requirements for mixed workload for a cluster consisting of 4 instances.

dundant ones. As new KV caches are created, they are flagged for replication in the second instance, and redundant KV lines are updated. After prefill, the first instance returns to decoding, with the scheduler balancing workload by equalizing batch size and request length. Among available pairs, the one with the most free space handles the next prefill. If multiple requests arrive simultaneously, they are evenly distributed among pairs to minimize TTFT. To address inter-pair imbalances, the manager incrementally builds redundancy across pairs, given sufficient interconnect bandwidth and memory capacity.

4.2.3 Instance

Each AcceLLM instance comprises four accelerators and the open-source vLLM library (version 0.4.2) (Kwon et al., 2023) is utilized to manage inference requests within the instance. Tensor parallelism is configured to four, and batching is employed to minimize redundant memory reads of model weights and to enhance the utilization of available accelerator computational resources (Yu et al., 2022; Dao et al., 2022). Within each instance, the vLLM code has been modified to prevent the batching of prefill and decoding operations at any point ensuring that no latency spikes occur during the decoding phase.
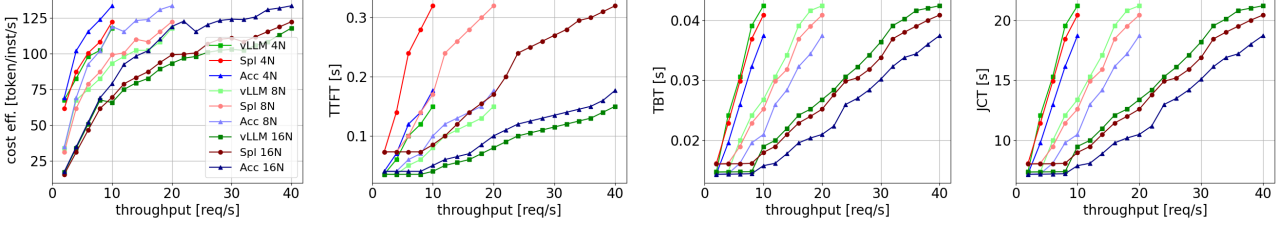
Figure 11. Latency and cost efficiency for the mixed workload with simulated Nvidia H100 instances.
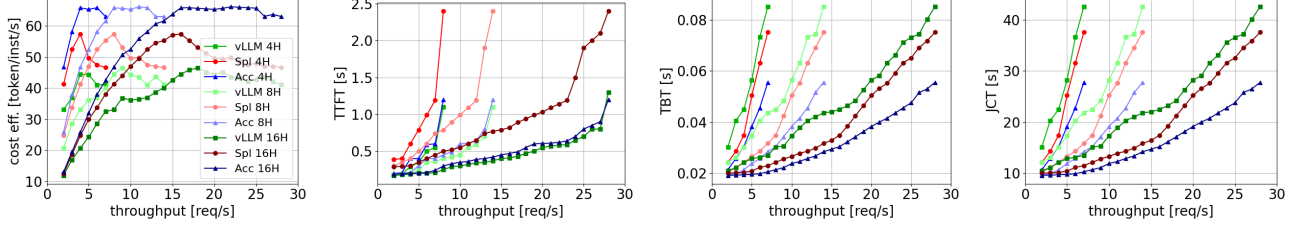

Figure 12. Latency results for the mixed workload with Ascend 910B2 instances.

4.2.4 KV Cache Transfer

The NVlink connection between instance accelerators enables efficient KV cache migration, essential for phased disaggregated computation. Instead of waiting until the prefill phase ends which would lead to very high serving latency, KV caches are transferred per layer as they are generated, reducing JCT latency, especially for large prompts. During each layer in the prompt and decoding phases, KV caches are transferred to the paired instance, allowing computation to proceed seamlessly. The vLLM code is modified to enable these per-layer transfers and to concatenate request tensors, ensuring minimal latency overhead due to NVlink's high-speed interconnect.

4.2.5 Insufficient Memory Capacity for Redundancy

When high request volumes prevent full data replication across instances, one instance is designated to handle both prefill and decoding (though not simultaneously), while the other three operate in decoding-only mode. These three instances overwrite redundant KV caches with new decoding requests, thus serving a greater number of requests. The dual-phase instance retains about one-third of the KV caches of each decoding instance as redundant copies. When no new requests are incoming, decoding is distributed across all four instances, improving load balancing.

5 EVALUATION

5.1 Inference System Modeling

Assessing AcceLLM in an actual cluster environment requires significant resource allocation. Therefore, to evaluate AcceLLM's performance on a cluster level design, we have developed a simulator that faithfully simulates the computation, HBM bandwidth, memory requirements and KV cache transfer costs of our LLM inference system. For the performance model, we leverage extensive data from experiments performed on the Huawei Ascend 910B2 accelerator and simulated Nvidia's H100 SXM5 GPU whose device specifications are shown in Table 1.

5.2 Workload-Baseline

We perform our evaluations based on the Llama-2 70B parameter model (Touvron et al., 2023). For evaluating the experiments, three distinct workloads were tested. The token number for these workloads is shown in Table 2, with every request drawn from a uniform distribution.

The first baseline for comparing AcceLLM 's performance is Splitwise (Patel et al., 2023), which also applies a state-of-the-art disaggregation policy between prefill and decoding phases, similar to (Zhong et al., 2024; Strati et al., 2024b; Hu et al., 2024). Splitwise typically allocates most instances to prefill, though most compute time is spent on token generation (Griggs et al., 2024; Nie et al., 2024). In our simulations, we prioritize decoding for Splitwise, batching both phases under high loads, and exclude non-disaggregated instances to avoid latency spikes.

Our second state-of-the-art baseline is vLLM (Kwon et al., 2023), which uses sequence parallelism for memory efficiency. We simulate Splitwise and vLLM with 4, 8, and 16 instances, applying the same inter-accelerator optimizations as AcceLLM. For Splitwise, 1, 2, and 4 instances handle prefill in the 4, 8, and 16-instance cases, respectively, while the rest focus on the more compute-intensive token generation phase.
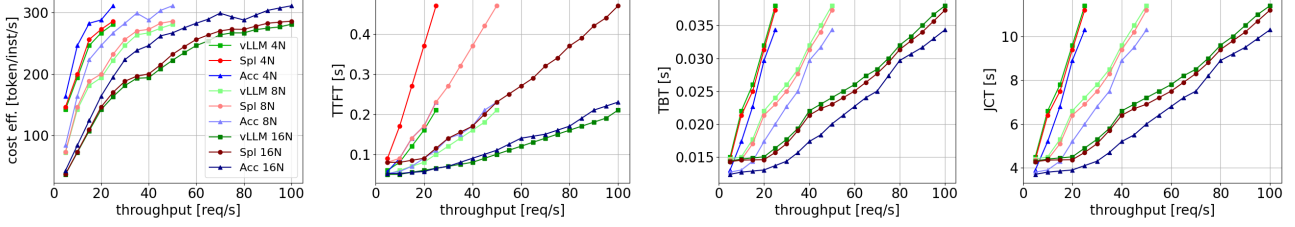
Figure 13. Latency results for the light workload with simulated Nvidia H100 instances.
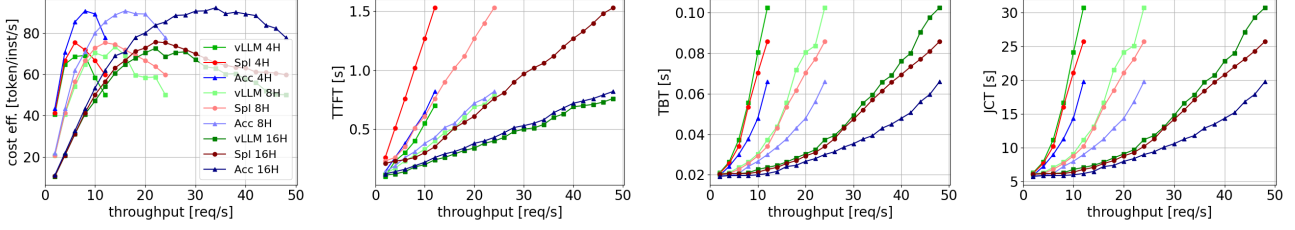

Figure 14. Latency results for the light workload with Ascend 910B2.

5.3 Experimental Results

Impact of Device Memory Capacity

Creating KV cache replicas for serving requests requires additional memory space in instances compared to other state-of-the-art methods. Figure 9 illustrates the amount of memory needed per instance for AcceLLM, Splitwise and vLLM in order to reach peak performance in terms of cost efficiency and JCT. It demonstrates that AcceLLM requires 1, 3.5, and 5 GB more memory than Splitwise and vLLM for handling 4, 8, and 12 requests per second, respectively, to achieve superior performance. However, in these three cases, instance memory is never saturated in either system leaving enough space for redundancy. Additionally, in many serving systems, certain JCT requirements have to be met, meaning that the instances' memory is never fully utilized, leaving sufficient space for the redundant KV cache copies (Liu et al., 2024; Griggs et al., 2024).

Impact of Interconnect Bandwidth

Moving KV caches between instances requires a certain interconnect bandwidth to achieve peak performance. Figure 10 illustrates the impact of interconnect speed on token throughput and JCT time. Both AcceLLM and Splitwise reach peak performance at approximately the same interconnect speed. The primary bulk of KV cache movement occurs during the prefill computation. In AcceLLM, additional KV cache line movements occur during decoding to keep the replicated caches updated. These extra movements are minimal compared to those during prefill and have a limited effect on overall interconnect utilization. We exclude vLLM from the figure because its system does not require KV cache transfers between instances.

AcceLLM Cost Efficiency

Considering the mixed workload case, in Figure 11(a) and Figure 12(a), we observe that across all clusters, the amount of tokens generated per instance per second follows a linear increase until it saturates and then declines. Initially, the linear growth is attributed to the positive impact of increasing the batch size on token generation. However, beyond a certain point, clusters struggle to manage this increased batch size resulting from the rise in incoming requests per second. Consequently, there is a subsequent decline in performance as the clusters' computation capacity is already saturated.

Load Balancing With Varying Request Volumes

AcceLLM demonstrates superior capability in handling a larger volume of incoming requests per second before reaching computational resource saturation (Figure 11(a) and Figure 12(a)), compared to Splitwise and vLLM. It outperforms Splitwise and vLLM by nearly 30% in terms of the amount of tokens generated per instance per second, resulting in up to a 30% reduction in JCT (Figure 11(d) and Figure 12(d)). This improvement is attributed to AcceLLM's dynamic load balancing of requests across instances during the decoding phase, facilitated by redundant KV caches. With the redundant KV caches, AcceLLM evenly distributes batched requests per instance regardless of their token generation length. Additionally, this reduction in JCT comes from the seamless conversion of instances from the prefill phase to the decoding one and vice versa, ensuring no instance remains idle at any given time. Conversely, in Splitwise, dedicating a large number of instances to prefill tasks often results in significant periods of resource idleness when there are no incoming requests, leading to resource wastage and a significant latency increase in JCT compared to AcceLLM.
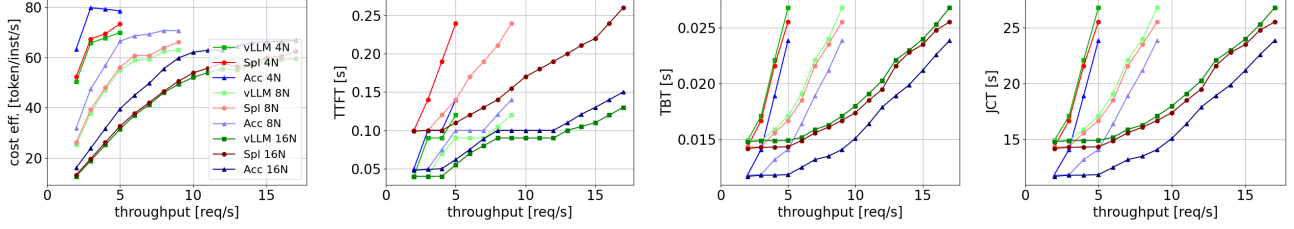
9

Figure 15. Latency results for the heavy workload with H100 instances.

Load balancing and the use of dynamic instances also contribute to a reduced TBT (Figure 11(c) and Figure 12(c)). These approaches allows more instances to share the computational load, ensuring a nearly equal distribution of workload across resources. In the prefill phase, AcceLLM achieves prompt computation in almost half the time compared to Splitwise (Figure 11(b) and Figure 12(b)), facilitated by its ability to dynamically allocate double the number of instances for prefill tasks when necessary. Moreover, it computes the prefill phase in about the same time as in vLLM which batches together prefill and decoding computation.

It is worth noting that in Figure 12(b), when the incoming request rate increases substantially (6, 12, and 24 requests per second for systems with 4, 8, and 16 instances respectively), Splitwise queues incoming requests before prefill computation because the previous ones have not completed their prefill computation, resulting in additional latency in serving requests. This issue is not observed in AcceLLM, as more instances can be allocated dynamically for prefill when needed. Allocating more instances to prefill in Splitwise could reduce queuing effects but would negatively impact JCT, as fewer instances would be available for the more time-intensive decoding phase.

Load Imbalance in Previous Work

Similar to the mixed workload scenario, AcceLLM consistently outperforms Splitwise and vLLM across all metrics in the light workload case, with the exception of TTFT for vLLM. Looking at the TTFT metric (Figure 13(b)) for the system with H100 accelerators, we observe that in Splitwise, prefill computation takes nearly 0.45 seconds in the worst-case scenario. This implies that prefill instances are active for only about 0.45 seconds per second, leaving them idle for the remainder of the time. However, in our AcceLLM system, all instances are utilized continuously, without idle periods which leads to the faster serving of requests (Figure 13(d)).

Overloading Prefill Instances

On the other hand, in Huawei's cluster (Figure 14(b)), significant queuing of new incoming requests occurs in the
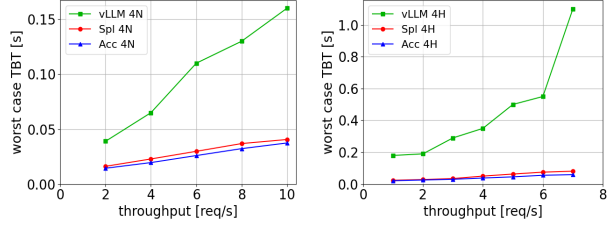

Figure 16. Worst case TBT latencies.

Splitwise system due to the insufficient number of instances handling prefill computation. In contrast, AcceLLM's capability to dynamically convert instances between phases helps mitigate these delays in computing new requests.

Importance of Load Balancing for Heavy Workloads

Figure 15 shows results under heavy workloads, where high KV cache demand reduces the rate of new requests handled per second. Effective load balancing of decoding requests becomes crucial in these scenarios, as increased token generation results in larger batch sizes, which can lead to imbalance across instances. By employing load balancing, AcceLLM overcomes this issue, serving requests up to 3 and 3.5 seconds faster than Splitwise and vLLM, respectively (Figure 15(d)).

Figure 16 illustrates the worst-case scenario for TBT latency. In vLLM, the prefill and decoding processes are batched together, resulting in some tokens being generated very slowly since they are computed alongside the incoming prompt. AcceLLM prevents these latency spikes by disaggregating the prefill and decoding phases, ensuring they are computed by separate instances.

6 CONCLUSIONS

We have presented AcceLLM, a novel methodology that employs redundant copies of LLM inference data to minimize latency and to ensure efficient allocation of hardware resources. Our methodology leverages dynamic distinct instances and achieves load balancing. Targeting simulated Nvidia H100 and Huawei Ascend 910B2 accelerators, we demonstrate significant gains of up to 30% in key user-experience metrics, sustaining negligible memory impact, compared to state-of-the-art methods.

REFERENCES

Agrawal, A., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., and Ramjee, R. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Agrawal, A., Kedia, N., Panwar, A., Mohan, J., Kwatra, N., Gulavani, B. S., Tumanov, A., and Ramjee, R. Taming throughput-latency tradeoff in llm inference with sarathi-serve. 2024.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P. S., Yang, Q., and Xie, X. A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3), mar 2024. ISSN 2157-6904. doi: 10.1145/3641289. URL https://doi.org/10.1145/3641289.

Dao, T. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

FasterTransformer. Faster transformer. https://github.com/NVIDIA/FasterTransformer. URL https://github.com/NVIDIA/FasterTransformer.

Griggs, T., Liu, X., Yu, J., Kim, D., Chiang, W.-L., Cheung, A., and Stoica, I. Mélange: Cost efficient large language model serving by exploiting gpu heterogeneity, 2024.

Holmes, C., Tanaka, M., Wyatt, M., Awan, A. A., Rasley, J., Rajbhandari, S., Aminabadi, R. Y., Qin, H., Bakhtiari, A., Kurilenko, L., et al. Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference. *arXiv preprint arXiv:2401.08671*, 2024.

Hong, K., Dai, G., Xu, J., Mao, Q., Li, X., Liu, J., Chen, K., Dong, Y., and Wang, Y. Flashdecoding++: Faster large language model inference on gpus. 2024.

Hu, C., Huang, H., Xu, L., Chen, X., Xu, J., Chen, S., Feng, H., Wang, C., Wang, S., Bao, Y., et al. Inference without interference: Disaggregate llm inference for mixed downstream workloads. *arXiv preprint arXiv:2401.11181*, 2024.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with pagedattention. In Flinn, J., Seltzer, M. I.,

Druschel, P., Kaufmann, A., and Mace, J. (eds.), *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165. URL https://doi.org/10.1145/3600006.3613165.

Li, J., Xu, L., Xu, H., and Akella, A. Blockllm: Multi-tenant finer-grained serving for large language models, 2024.

Lin, B., Peng, T., Zhang, C., Sun, M., Li, L., Zhao, H., Xiao, W., Xu, Q., Qiu, X., Li, S., Ji, Z., Li, Y., and Lin, W. Infinite-llm: Efficient llm service for long context with distattention and distributed kvcache, 2024.

Liu, J., Wu, Z., Chung, J.-W., Lai, F., Lee, M., and Chowdhury, M. Andes: Defining and enhancing quality-of-experience in llm-based text streaming services. *arXiv preprint arXiv:2404.16283*, 2024.

McColl, B. Superclouds: Scalable high performance non-stop infrastructure for ai and smart societies. In *International Conference on Smart Cities, Infrastructure, Technologies and Applications*, pp. 3–5. Springer, 2017.

Miao, X., Shi, C., Duan, J., Xi, X., Lin, D., Cui, B., and Jia, Z. Spotserve: Serving generative large language models on preemptible instances, 2023.

Nie, C., Fonseca, R., and Liu, Z. Aladdin: Joint placement and scaling for slo-aware llm serving, 2024.

Patel, D. and Ahmad, A. Large Language Model cost analysis,, 2024. URL https://www.semianalysis.com/p/the-inference-cost-of-search-disruption.

Patel, P., Choukse, E., Zhang, C., Goiri, Í., Shah, A., Maleki, S., and Bianchini, R. Splitwise: Efficient generative llm inference using phase splitting. *arXiv preprint arXiv:2311.18677*, 2023.

Recasens, P. G., Zhu, Y., Wang, C., Lee, E. K., Tardieu, O., Youssef, A., Torres, J., and Berral, J. L. Towards pareto optimal throughput in small language model serving. *arXiv preprint arXiv:2404.03353*, 2024.

Schillaci, Z. Llm adoption trends and associated risks. In *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*, pp. 121–128. Springer Nature Switzerland Cham, 2024.

Sheng, Y., Zheng, L., Yuan, B., Li, Z., Ryabinin, M., Chen, B., Liang, P., Ré, C., Stoica, I., and Zhang, C. Flexgen: high-throughput generative inference of large language models with a single gpu. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Srivatsa, V., He, Z., Abhyankar, R., Li, D., and Zhang, Y. Preble: Efficient distributed prompt scheduling for llm serving. 2024.

Strati, F., Ma, X., and Klimovic, A. Orion: Interference-aware, fine-grained gpu sharing for ml applications. In *Proceedings of the Nineteenth European Conference on Computer Systems*, EuroSys '24, pp. 1075–1092, New York, NY, USA, 2024a. Association for Computing Machinery. ISBN 9798400704376. doi: 10.1145/3627703.3629578. URL https://doi.org/10.1145/3627703.3629578.

Strati, F., Mcallister, S., Phanishayee, A., Tarnawski, J., and Klimovic, A. Dejavu: KV-cache streaming for fast, fault-tolerant generative llm serving. *arXiv preprint arXiv:2403.01876*, 2024b.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Wu, B., Liu, S., Zhong, Y., Sun, P., Liu, X., and Jin, X. Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism, 2024.

Yu, G., Jeong, J. S., Kim, G., Kim, S., and Chun, B. Orca: A distributed serving system for transformer-based generative models. In Aguilera, M. K. and Weatherspoon, H. (eds.), *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022*, pp. 521–538. USENIX Association, 2022. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Zhao, X., Jia, B., Zhou, H., Liu, Z., Cheng, S., and You, Y. Hetegen: Heterogeneous parallel inference for large language models on resource-constrained devices, 2024.

Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and Zhang, H. Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving. *arXiv preprint arXiv:2401.09670*, 2024.