ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production

Yuxing Xiang[1,2] Xue Li[2] Kun Qian[2] Yan Zhang[1]
Wenyuan Yu[2] Ennan Zhai[2] Xin Jin[1] Jingren Zhou[2]
[1]*School of Computer Science, Peking University* [2]*Alibaba Group*

Abstract. With the widespread adoption of Large Language Models (LLMs), serving LLM inference requests has become an increasingly important task, attracting active research advancements. Practical workloads play an essential role in this process: they are critical for motivating and benchmarking serving techniques and systems. However, the existing understanding of real-world LLM serving workloads is limited due to the lack of a comprehensive workload characterization. Prior analyses remain insufficient in scale and scope, thus failing to fully capture intricate workload characteristics.

In this paper, we fill the gap with an in-depth characterization of LLM serving workloads collected from our worldwide cloud LLM serving service, covering not only language models but also emerging *multimodal* and *reasoning* models, unveiling important *new* findings in each case. Moreover, based on our findings, we propose ServeGen, a principled framework for generating realistic LLM serving workloads by composing them on a per-client basis. Practical use cases validate that ServeGen achieves more accurate performance benchmarking compared to naive workload generation, and reveals new design implications that could otherwise be overlooked. ServeGen is open-sourced at `https://github.com/alibaba/ServeGen`.

1 Introduction

In recent years, the rapid evolution of Large Language Models (LLMs) [4, 5, 9, 19] has enabled fundamentally new applications, with large-scale deployment in production clusters serving substantial user traffic every day [7]. To accommodate this growing demand, a large body of research has focused on optimizing LLM serving in terms of model serving latency [25, 50], resource utilization [12, 23, 32], service quality [34, 57], and beyond [24, 49, 56].

Inference serving workloads play an important role in this innovation process: they motivate the design of new optimization techniques and systems, and the effectiveness of the latter must be validated under respective workloads. Yet, there is an absence of comprehensive, production-scale characterization of real-world serving workloads. The status quo is a mixture of (i) adapted workloads from general deep-learning or cloud computing tasks [27–29, 40, 47] (*e.g.,* using function invocations in serverless workloads as inference requests), and (ii) optimization- [16, 23, 37, 42] or pattern-specific [38, 46] analyses (*e.g.,* detailing only certain patterns), which remain insufficient in scale and scope.

The lack of practical workload characterization poses two obstacles to the innovation process of LLM serving systems. First, the many *uncharacterized* aspects of real-world workloads hinder new insights and motivations, especially for emerging scenarios such as serving multimodal [17, 53] and reasoning [2, 20] models. Second, even for serving normal (*i.e.,* non-reasoning) language models that have been extensively studied, the inadequate understanding of real-world workloads may still result in *unrealistic* benchmarking when evaluating emerging optimizations. The *de facto* approach (referred to as NAIVE) adopted by many studies [33, 49, 50, 55] generates workloads by simply combining certain arrival *traces* (*e.g.,* sampled from Poisson or Gamma processes, or scaled from published traces [40]) with *datasets* (*e.g.,* ShareGPT [6]).[1] However, prior experience in cloud workload modeling [15, 43] has highlighted more intricate workload patterns, such as "heterogeneity" [39] and "imbalance" [35], revealing that "naively-generated workloads are misleadingly easier to serve than real historial ones" [15]. In practice, scaling serving optimizations to deployment has been met with unforeseen difficulties, such as performance degradation [30] and major revisions in system design [1].

As a large cloud inference service provider, we aim to fill this gap with an extensive and detailed characterization of real-world LLM serving workloads, analyzing a diverse range of models (see Table 1) and billions of requests collected from our production clusters over four months. We provide a comprehensive analysis of LLM serving workloads that covers language (§3), multimodal (§4), and reasoning (§5) workloads, unveiling important *new* findings. We release a principled framework, ServeGen, which allows practitioners to incorporate our findings and generate realistic workloads that better reflect system performance compared to NAIVE workload generation (§6), thus facilitating the motivation and evaluation of ongoing research.

Characterizing language model workloads. We begin with a characterization of various (non-reasoning) language model workloads based on their arrival patterns (§3.1) and input/output lengths (§3.2). While there is prior work analyzing language model workloads, our analysis yields important new findings: (i) request arrivals exhibit a complex bursty pattern that goes beyond any single stochastic process (*e.g.,* a

[1] Prior work has used the terms "trace", "dataset", and "workload" interchangeably. In our discussion, "trace" denotes request arrival timestamps, while "dataset" refers to request data distributions exclusively.

Table 1. The list of workloads and models in our study.

Category	Name	Model	Description	Workload Information
Language	`M-large`	General model (310B)	Largest, general-purpose	240M requests (one month)
	`M-mid`	General model (72B)	Balanced, general-purpose	2.1B requests (one month)
	`M-small`	General model (14B)	Cheapest, general-purpose	767M requests (one month)
	`M-long`	General model (72B, 10M context)	Long-document comprehension	48M requests (one week)
	`M-rp`	Domain-specific model	Role-playing	49M requests (one week)
	`M-code`	Domain-specific model	Code completion	276M requests (one week)
Multimodal	`mm-image`	Qwen2.5-VL-72B	Image & text input	28M requests (one month)
	`mm-audio`	Qwen2-Audio-7B	Audio & text input	420K requests (one month)
	`mm-video`	Qwen2.5-VL-72B	Video & text input	1.2M requests (one month)
	`mm-omni`	Qwen2.5-Omni-7B	Omni-modal input	8.7M requests (one week)
Reasoning	`deepseek-r1`	deepseek-r1-671B	Full reasoning model	14.0M requests (one week)
	`deepqwen-r1`	deepseek-r1-distill-qwen-32B	Distilled reasoning model	4.8M requests (one week)

gamma process is not necessarily the best fit in all cases); and (*ii*) the input/output length distributions can be modeled by combinations of classic distributions, but the corresponding parameters vary significantly over time. Considering these findings, we conduct a deep-dive analysis by decomposing the workloads by clients (§3.3), revealing a *causal modeling* of real-world workloads: most nondeterministic patterns in request arrivals (*e.g.,* bursts) and length distributions (*e.g.,* high dynamics over time) are caused by several top clients, while the behaviors of most clients remain stable and predictable. This finding is valuable for generating realistic workloads.

Characterizing multimodal and reasoning workloads. We also analyze inference serving workloads of emerging multimodal and reasoning models, highlighting their unique characteristics. For multimodal workloads, we report significant load variance across modalities (§4.1) and substantial request heterogeneity (§4.2), unveiling inefficiencies in the prefill phase of LLM inference. For reasoning workloads, the long and bimodal distribution of reasoning lengths (§5.1) and the more stable arrival pattern from multi-turn conversations (§5.2) present both challenges and opportunities in optimizing the decoding phase. In both scenarios, similarly, we analyze the workloads with client decomposition (§4.3 and §5.3) to deepen our characterization, again capturing the diverse patterns through causal modeling.

Workload generation. While the aforementioned findings help motivate the design of next-generation LLM serving systems, it remains crucial for practitioners to be able to evaluate said systems with realistic workloads. However, full-scale production workloads are not always available due to privacy concerns, particularly in emerging serving scenarios. Moreover, the few published workloads [16, 23, 37, 42, 46] are limited to a specific scale and are subject to the so-called "workload churn" [14]. To better share our insights and further facilitate the community, we build and release ServeGen, a workload generation framework to generate realistic serving workloads. ServeGen performs principled modeling of workloads on a *per-client* basis based on our findings to generate realistic workloads, and is easy to use (§6.1). Our evaluation shows that ServeGen outperforms the NAIVE workload generation approach by producing workloads that better align with real ones (§6.2). Additionally, we demonstrate that ServeGen is beneficial for benchmarking and motivating better serving system designs in production via case studies on instance provisioning and PD-disaggregation [57].

Contributions. Our main contributions are as follows.

- We provide a comprehensive study of real-world LLM serving workloads in a large-scale production environment, which not only covers language models, but also emerging multimodal and reasoning models.
- We characterize production-level LLM serving workloads and conduct in-depth analysis by client decomposition, revealing important new findings.
- We release ServeGen, a principled framework for generating realistic serving workloads based on our findings to help motivate and benchmark future research.

2 Background

2.1 LLM Basics

Basic LLM inference. The typical inference workflow for an LLM request comprises two key phases: *prefill* and *decoding*. In the prefill phase, all *input* tokens in the user prompt are processed to generate the first *output* token. Subsequently, the decoding phase auto-regressively generates the rest of the output tokens sequentially, until either the generation of an end-of-sequence (EOS) token or a predefined maximum output length is reached. In both phases, requests are commonly batched [54] and processed simultaneously to enhance the serving throughput. Consequently, the arrival pattern and input/output lengths of requests are strongly relevant to LLM inference performance, as they impact the batching result and computational load during execution.

Multimodal models. Multimodal LLMs [17,53] are extended with the ability to process and integrate multiple types of data beyond text prompts, such as images, audio, and video, al-

Table 2. Comparison between our work and prior characterizations of LLM serving workloads. Dashes indicate unavailable data.

	Ours	BurstGPT [46]	LMM [38]
Characterization ▷ Scale			
Duration	4 months	4 months	2 days
#Models	12	2	-
#Requests	3.54B	5.29M	-
Characterization ▷ Scope			
Workloads	Language, Multi-modal, Reasoning	Language	Image-modal
Patterns	Variant burstiness Distribution shifts Conversations	Variant burstiness	Image data distribution
Workload Generation			
Approach	Parameterized clients	Parameterized burstiness	NAIVE

lowing for richer user interactions. In a typical multimodal inference workflow, a model must first process its multimodal inputs through a series of *downloading* (fetching data from URLs), *normalizing* (*e.g.,* resizing images or resampling audio), and *encoding* (through modality-specific adapters, such as ViT [21]) stages to obtain their embeddings, which are fused with the text embeddings. The inference then proceeds in a process identical to basic LLM serving. As such, multimodal data distributions play a crucial role in the inference performance of multimodal LLMs.

Reasoning models. A significant recent progress in LLMs is the rise of reasoning [2, 20], which shows remarkable capabilities in complex coding, math, and problem-solving tasks. These models' output tokens are divided into two sections: first the *reason* tokens, where the model performs test-time computation [3], and second the *answer* tokens that actually answer the input prompt. This behavior makes reasoning workloads stand out from normal language model workloads, altering the workload statistics (*e.g.,* longer outputs) while also potentially enabling new optimizations.

2.2 LLM Serving Workloads

Workload characterization and generation. Optimization of LLM serving systems promises significant performance gains and substantial cost reductions. However, achieving this goal requires a deep understanding of real-world workloads, which is often unavailable due to the absence of a comprehensive production-scale workload characterization. Table 2 summarizes related work on LLM inference workload characterization, omitting various brief analyses found in other work [16, 23, 37, 42] that are optimization-specific and more limited in scope. As shown by the comparison, state-of-the-art characterizations are lacking in terms of scale, and leave many workload patterns *uncharacterized*. Furthermore, this inadequacy results in *unrealistic* workload generation approaches, restricting practitioners to workloads that cannot fully capture real-world patterns. Thus, we are motivated to perform a more extensive characterization of real LLM serving workloads in production. We then share our insights by building and releasing ServeGen, a principled framework for generating realistic serving workloads to foster future research.

Workload source. Alibaba Cloud Model Studio is a cutting-edge AI model service platform that enables users to build and use various custom model services. It hosts over 200 foundation models and thousands of fine-tuned models, supports applications deployed by hundreds of enterprises, and serves millions of requests each day. To support such a high and diverse workload, Model Studio maintains O(10K) GPUs distributed in dozens of regions and zones, making it a worldwide large model service platform.

Our characterization is supported by real inference workloads running in Model Studio. The analyzed workloads span four months, containing 12 models and billions of requests from datacenters in different geolocations, as shown in Table 1. We source request metadata from our log store for the backend inference engines, collecting detailed information including request arrival and execution times, payload (*e.g.,* input and output lengths, chat histories, and multimodal inputs), and other relevant data, all sanitized to respect client privacy. Notably, these data are agnostic of serving system internals or implementation details (as supposed to metrics like request completion time), allowing for an unbiased and comprehensive understanding of LLM serving workloads.

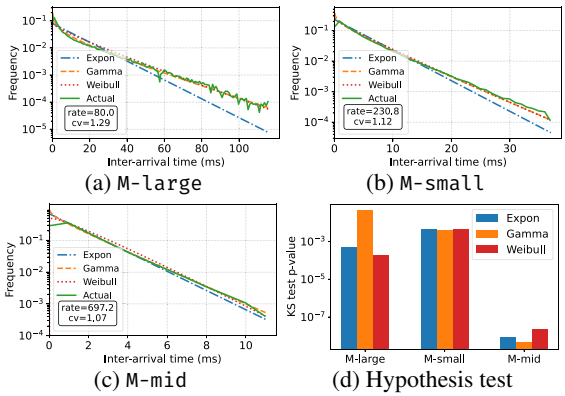
3 Characterizing Language Workloads

This section analyzes language model workloads listed in Table 1. We report findings for arrival times (§3.1) and input/output length distributions (§3.2), two essential traits that affect the performance of LLM serving systems. Importantly, we show that much of the complex underlying patterns behind our findings can be explained by client decomposition (§3.3).

3.1 Request Arrival Pattern

Bursty short-term arrival patterns. Figure 1 characterizes the inter-arrival time (IAT) distributions for `M-large`, `M-small`, and `M-mid` within a 20-minute window. Conforming to existing analyses [40, 46], we find that the arrival patterns exhibit notable *burstiness*, indicated by coefficients of variation (CVs) greater than 1. Consequently, Poisson processes (CV=1) often poorly model bursty workloads (such as in Figure 1(a)), where Gamma and Weibull processes are better alternatives. However, there is not a universally best modeling, which is validated in Figure 1(d), where we apply the Kolmogorov-Smirnov (KS) test to check whether the IATs came from Exponential, Gamma, or Weibull distributions.[2] None of the distributions has the largest p-value consistently, indicating variable goodness of fit. In fact, the best-fit

[2]Indeed, these p-values are too small to deny the null hypothesis (that the arrival is modeled by some distribution), which is common for the KS test when the sample size is large. However, comparing p-values remains helpful.



(a) `M-large` (b) `M-small` (c) `M-mid` (d) Hypothesis test

Figure 1. Inter-arrival time characterization.

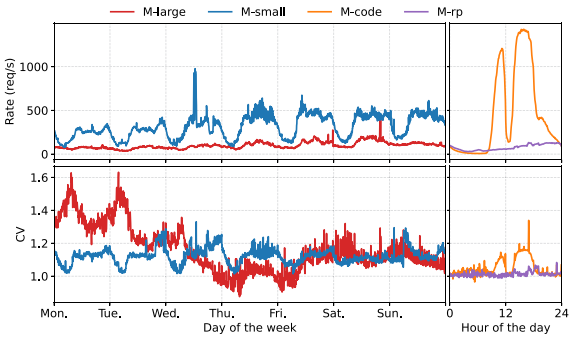


Figure 2. Long-term rate and CV shifts.

choices are different across workloads: Gamma for `M-large`, Weibull for `M-mid`, and even Exponential can be a good fit for `M-small`. Practically, this implies that arrival patterns in real-world workloads should be modeled flexibly using different distributions to better preserve their characteristics.

> **Finding 1**: The short-term arrival of LLM requests is often bursty (CV > 1), exhibiting complex patterns beyond any single stochastic process.

Shifting rate and burstiness. Figure 2 depicts the request rate and CV computed in 5-minute windows for multiple workloads, ranging from general-purpose (over a week) to task-specific ones (over a day). We observe evident diurnal fluctuations for the arrival rate: the load peaks during the afternoons while dropping significantly in the early mornings, resulting in potentially extreme rate shifts (as shown for `M-code`). Moreover, Figure 2 also displays diverse and shifting CV patterns for different workloads, underscoring the instability of burstiness [46] in real-world workloads. For instance, `M-large` was continuously bursty for two days (Mon. and Tue.) before turning stable (Thu. and Fri.). Meanwhile, request arrivals in `M-rp` remain non-bursty for the entire day of analysis. We believe such diversity is partly caused by the invocation pattern associated with each workload: while role-playing (`M-rp`) typically involves human interaction (*i.e.,* invoked via chatbots), where bursts are less common, general-purpose workloads (`M-large`) likely include API invocations with bursts of batched request submission.

These shifts in rate and burstiness have strong implications for LLM serving system design. For one, rate shifts demonstrate the importance of *auto-scaling* mechanisms in order to properly provision resources. For another, CV shifts provide both challenges and opportunities for designing *request scheduling* policies, which should acknowledge and adapt to different levels of burstiness. In contrast, systems that assume static workload patterns may not perform well in practice.

> **Finding 2**: The arrival of LLM serving requests shows a diverse shifting pattern in terms of rate and burstiness, calling for adaptive system design.

3.2 Input and Output Length Distribution

We now characterize the input and output lengths of requests by examining their distributions in Figure 3. Figure 4 further presents the correlation between input and output lengths by binning similar input lengths and showing the 90% percentile range and median of the respective output lengths.

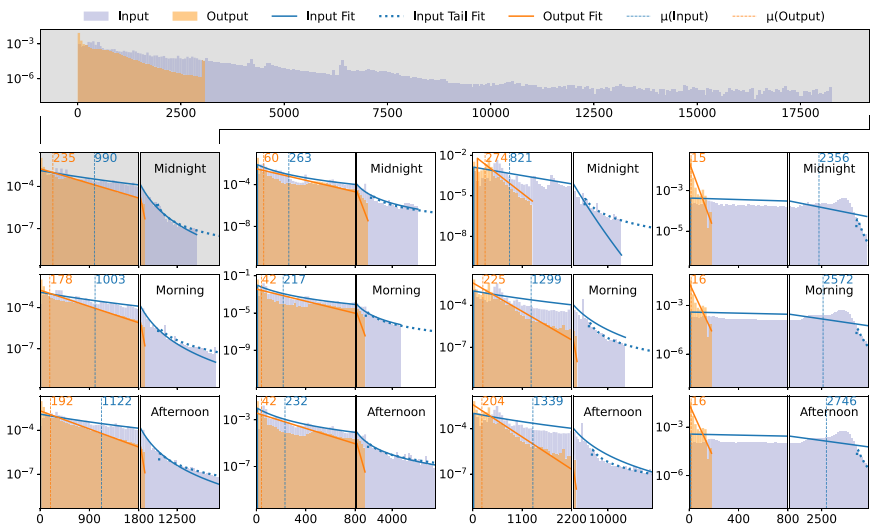
Modeling length distributions. Existing studies [46] have advocated modeling request input lengths with the Zipf distribution, acknowledging an implicit power law: input lengths have large variance and a long upper tail (*i.e.,* requests with exceedingly long prompts). In our analysis, we find that input lengths in general-purpose workloads are best modeled by *Pareto* distributions mixed with *Log-normal* distributions (both are also power-law distributions) for handling the fat tail, as shown in Figure 3(a) and 3(b) with the *Input Fit* and *Input Tail Fit* curves. For task-specific workloads, the aforementioned model is less accurate due to domain-specific bias, such as the usage of common system prompts or templates.

Surprisingly, we find that *Exponential* distributions fit remarkably well for output lengths, with the only obvious exception of `M-small` in Figure 3(b). While it is difficult to pinpoint the exact reason behind this phenomenon (likely a combined result of training and workload semantics), the implication is worth noting: the remaining output length of an LLM request is not conditioned on the generated length so far, *i.e.,* the output length distribution is *memoryless*.

Lastly, while Figure 4 exhibits a rough positive correlation between input and output lengths (*i.e.,* long prompts lead to long responses), the relation is not as pronounced as reported in previous studies [46]. We believe that in practice, the correlation is diminished by complicated workload semantics, such as prompt templates or structured outputs.

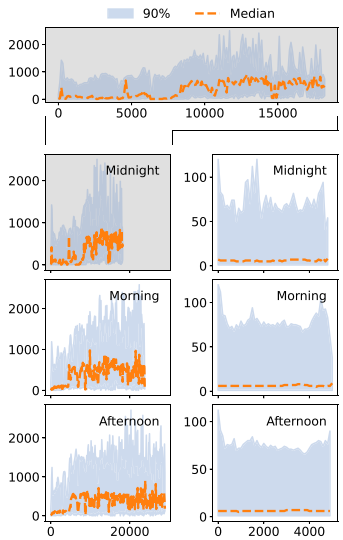
> **Finding 3**: The input length distribution can be modeled with a mixture of Pareto and Log-normal distributions, and the output with Exponential distributions. Correlation between input and output lengths is weak.

Shifting length distributions. Motivated by Finding 2, we repeat the preceding analysis using data sampled from three



(a) M-mid (b) M-small (c) M-long (d) M-code

Figure 3. Input and output length distribution. *x-axis*: # tokens; *y-axis*: frequency. Each subfigure corresponds to a specific workload and time period, split to two consecutive x-scales to show both the shift in average lengths (left) as well as the tail distribution (right).



(a) M-mid (b) M-code

Figure 4. Input and output length correlation. *x-axis*: input length; *y-axis*: output length.

different periods in a day, as shown in Figure 3 and 4. While the correlation appears independent of time, the actual distribution, contrary to common beliefs, *does* shift with time. Notably, the range of such shifts can be up to 1.63× for input (Figure 3(c)) and 1.46× for output (Figure 3(d)), measured by the maximal average length over the minimal.

Further, input and output length shifts occur *independently*, as demonstrated by `M-mid` in Figure 3(a): from *Midnight* to *Afternoon*, `M-mid`'s input length increases by 13% on average, while its output length drops by 18%. Intertwined with the request rate shifts, this observation translates to diverse load on the prefill and decoding phases of LLM serving, thus directly impacting system performance. Non-disaggregated serving systems may face variable performance interference between the two phases [57], while disaggregated systems might require phase-independent resource auto-scaling.

Finding 4: The input and output length distributions shift dynamically and independently over time, leading to diverse load fluctuations for prefill and decoding.

3.3 Client Decomposition

Thus far, we have uncovered several shifting patterns in LLM serving workloads with concrete real-world implications. These patterns are non-trivial to model because they are the aggregate behavior of multiple *clients*, each corresponding to an end user or upstream application (*e.g.,* a chatbot that relies on our service). For more insights, we conduct a decomposition analysis of the `M-small` workload on a *per-client* basis. We report substantial heterogeneity and stability in client behaviors, and further reveal that the shifting patterns are largely attributable to rate fluctuations among top clients.

Client heterogeneity and stability. Figure 5 characterizes client behaviors in terms of their rate, burstiness, and request length distribution, using the first 48-hour data from Figure 2. We observe highly skewed client rates: out of 2,412 clients, the top 29 clients (ranked by their rate in descending order) are responsible for 90% of the requests. Furthermore, client burstiness and input/output lengths span a diverse range, indicating the fundamental heterogeneity of clients.

Meanwhile, in isolation, top clients exhibit notable stability in all aspects other than their request rate, as shown in Figure 6. For example, the burstiness of Clients B, C, and D remains mostly stable within 48 hours, and the burstiness of Client A only deviates in the early mornings when the rate drops exceedingly low. Additionally, Clients A and D display stable input and output lengths, as indicated by the small error bars in the last-row subfigures, which visualize the range of average lengths during the entire period of our analysis.

Impact of top clients. These observations suggests the following *causal modeling*: characteristics of the whole workload are steered by a few top clients, whose rate fluctuations effectively cause the workload to shift towards different patterns.

This modeling indeed accounts for many previously found patterns in `M-small`. For example, note that in Figure 2, the workload temporarily bursts on Tuesday night. This matches the fact that in Figure 6, the rate of Client A (which is bursty) also sees a peak at around the same time. As for request lengths, the average input length of `M-small` decreases from *Midnight* to *Morning* in Figure 3, aligning with the increase of request rate from Client A (whose input lengths are shorter than average) from hour 1 to hour 9 in Figure 6. We rely on the same causal modeling to generate realistic workloads that encompass the intricate shifting patterns in §6, where we evaluate the accuracy and benefits of our approach quantitatively.

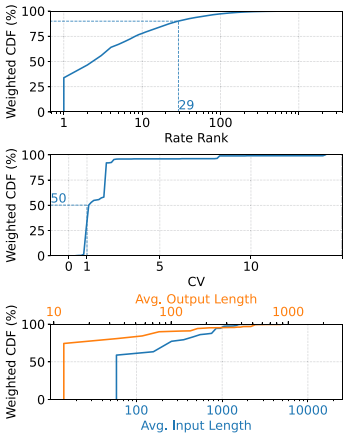


Figure 5. Client heterogeneity in terms of rate, *etc.* All CDFs are weighted by client rates.

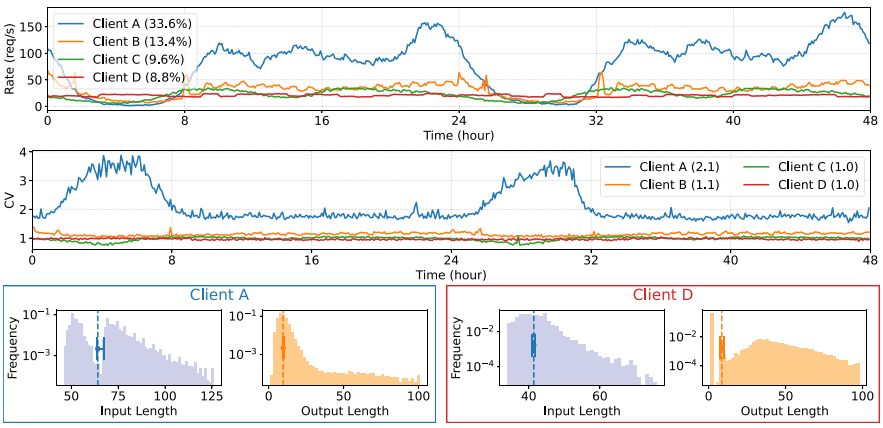


Figure 6. Characterization of the top four clients in the `M-small` workload in isolation, using the first 48-hour data from Figure 2. Vertical lines in the last-row subfigures indicate average input/output lengths, and error bars show the range of average lengths in 1-hour windows.

Finding 5: Real-world workloads consist of heterogeneous clients with skewed arrival rates. The top clients and their rate fluctuations largely explain the shifting workload patterns.

4 Characterizing Multimodal Workloads

We next examine workloads for multimodal models. Our characterization reveals that the tokenized length distributions are irregular across image, audio, and video modalities, contributing to highly variable modal load that also shifts over time (§4.1). Together with overhead from downloading, normalizing, and encoding (§4.2), multimodal inference is prone to considerable request *heterogeneity* between modalities, leading to prolonged time-to-first-token (TTFT). We report how client decomposition helps capture these patterns and facilitates a deeper understanding of multimodal workloads (§4.3).

4.1 Modality Load Variance

Load variance in different modalities. Figure 7 characterizes data distributions in `mm-image`, `mm-audio`, and `mm-video`, focusing specifically on the multimodal inputs. Unlike text prompts, multimodal inputs are more likely to have standard sizes depending on upstream applications. As such, in all three workloads, the tokenized lengths of multimodal inputs exhibit irregularly shaped distributions, clustering around certain values (*e.g.,* around 2,500 for `mm-video` in (b)) instead of following typical power-law distributions like the text modality (see Figure 3). In addition, given the diverse number of multimodal inputs per request (shown in (a)) and the lack of correlation between text and multimodal tokens (shown in (c)), we observe highly *varied* load on modality encoders, as illustrated in (d).

Two observations further complicate the load variance in multimodal workloads. (*i*) The variance of multimodal load can be independent of the textual load. For example, nine hours into the `mm-image` workload, the image token rate sees an abrupt increase, while the text token rate remains constant. (*ii*) Similar to the input/output distributions in language workloads, the multimodal data distributions also shift over time, as revealed in Figure 7(b). For instance, the average image length in `mm-image` varies by up to 19% over one day.

Load variance in omni-modality. Figure 8 presents the same analysis on `mm-omni`, an omni-modal workload where requests can contain more than two modalities. Unsurprisingly, the workload exhibits more complex variability, featuring a greater number of multimodal inputs per request and more diverse shifting patterns in input load (*e.g.,* audio load rises during the day, while image load becomes prominent past midnight). Moreover, as new applications of omni-modal LLMs emerge, we anticipate that the load variance in omni-modal workloads will continue to evolve.

In both cases, load variance presents challenges to the resource efficiency of multimodal inference, necessitating serving systems with more flexible resource scaling.

Finding 6: Multimodal data distributions exhibit irregular and independent shifts, underscoring significant load variance across modalities.

4.2 Request Heterogeneity

Multimodal inputs introduce complexity not only to the overall load, but also to individual requests. Figure 9 presents a breakdown of each request's input tokens in the `mm-image`, `mm-audio`, and `mm-video` workloads, revealing a flat distribution in every case. This indicates that real multimodal requests are *heterogeneous*, naturally ranging from text-heavy to multimodal-heavy in terms of input composition.

In practice, such heterogeneity is challenging for serving systems, as it translates into prolonged TTFT during inference, as shown in Figure 10. On one hand, for multimodal-heavy requests, the *download*, *normalization*, and *encoding* stages for tokenizing multimodal inputs all contribute to considerable extra overhead (reported in Figure 10(a)) in the first-token

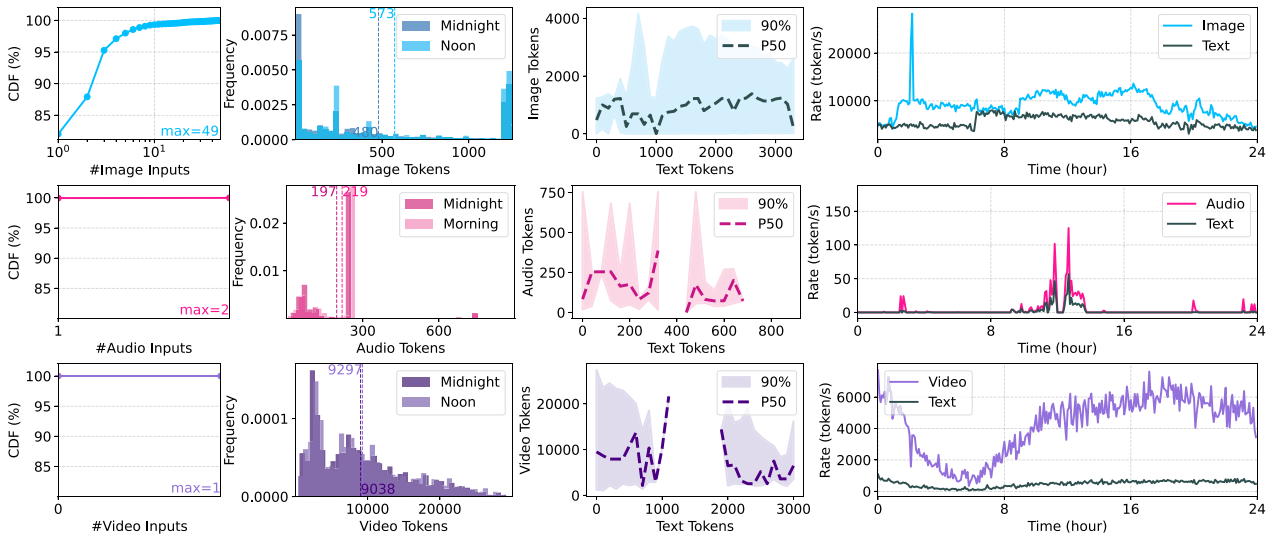


Figure 7. Characterization of multimodal inputs in `mm-image`, `mm-audio`, and `mm-video` (*Rows*). *Columns*: (a) #multimodal inputs per request; (b) length distribution of inputs; (c) correlation between text and multimodal tokens; (d) overall arrival rate of multimodal and text tokens.

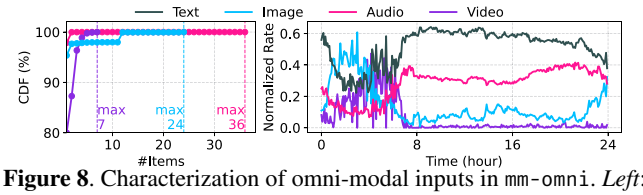


Figure 8. Characterization of omni-modal inputs in `mm-omni`. *Left*: number of multimodal inputs per request. *Right*: arrival rate of multimodal and text tokens, normalized by the total input rate.

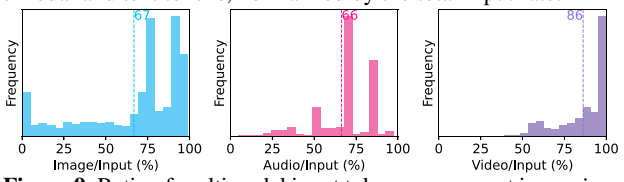


Figure 9. Ratio of multimodal input tokens per request in `mm-image`, `mm-audio`, and `mm-video`. Numbers indicate the average ratio.

generation process, directly lengthening the TTFT, as illustrated in Figure 10(b). For instance, half of the `mm-image` requests spend 75% of their TTFT before LLM prefilling. On the other hand, the extremely long-tailed distribution of encoder time in Figure 10 signifies potential queuing that affects text-heavy requests as well. For example, a request with few image tokens in `mm-image` may be blocked at the encoding stage by previously scheduled image-heavy requests; or it may experience slower encoding due to suboptimal batching that only considers prefill execution. This highlights the need for more advanced scheduling and batching strategies.

> **Finding 7**: Multimodal requests are heterogeneous with diverse ratios of multimodal inputs per request, leading to prolonged TTFTs that require tailored optimizations.

4.3 Mulitmodal Client Decomposition

Given the involved patterns in multimodal workloads, we present a client decomposition of `mm-image` similar to that in §3.3 to further complement our characterization. Notably, our causal modeling proposed by Finding 5 still applies, as we verify that load variance and request heterogeneity are explainable by the multimodal client behaviors.

Characterization of multimodal clients. Figure 11 summarizes the behaviors of 1,036 multimodal clients in `mm-image`, which are heterogeneous in terms of rate, burstiness, image length distributions, and image-to-input ratios per request. Interestingly, the last two CDFs concerning image data in Figure 11 exhibit a *staircase-like* pattern, hinting at the existence of text-heavy or multimodal-heavy clients.

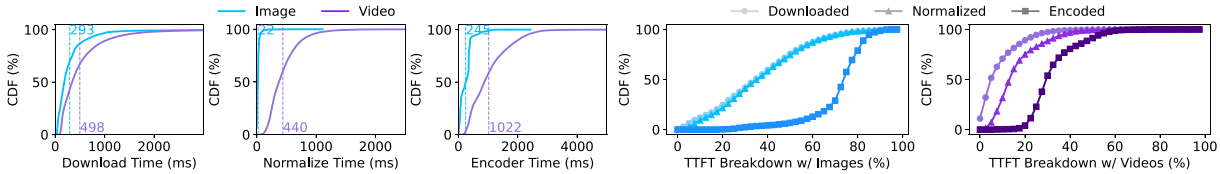
Indeed, some of the top clients show remarkably skewed data distributions, as represented by Client B in Figure 12, who exclusively sends images of the same size (around 1,200 tokens each) and requests that are similarly structured for the entire 24 hours during our measurement. In general, top-client behaviors remain stable and predictable, as indicated by the narrow error bars in the lower part of Figure 12.

Explaining workload patterns. We emphasize that the top clients presented in Figure 12 have a direct impact on the previously presented workload patterns. For one, the heterogeneity of clients directly contributes to the diverse image-to-input ratios across all requests. For another, Client B's rate ramps up nine hours into the workload, resulting in a surge of image-heavy requests (typical for this client) that exactly matches the increase of image load as shown in §4.1.

> **Finding 8**: Top clients in multimodal workloads exhibit diverse behaviors, and characterizing them helps explain the overall workload patterns.

5 Characterizing Reasoning Workloads

This section focuses on analyzing reasoning workloads. Our characterization shows that the unique "thinking" behavior



(a) Per-stage time during first-token generation. (b) CDF of cummulative time after each stage.

Figure 10. Breakdown of first-token time when serving requests with image or video inputs (`mm-image` and `mm-video`).

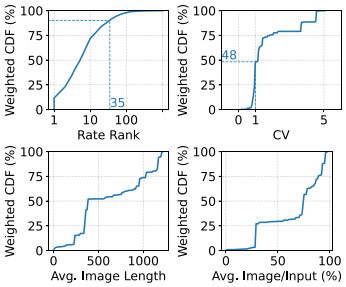


Figure 11. Clients in `mm-image`. CDFs are weighted by rates.

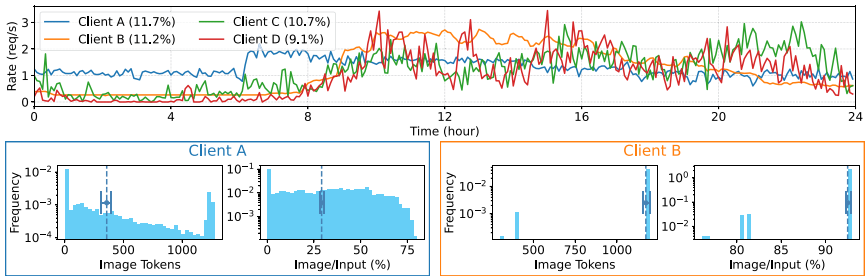
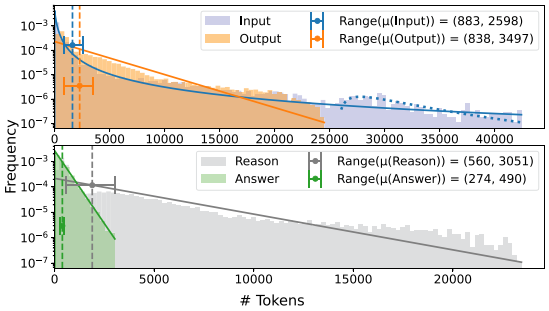
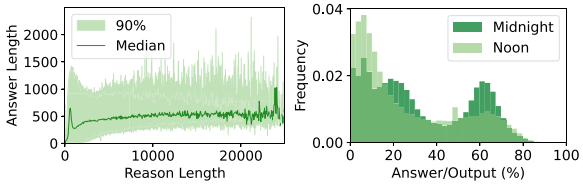


Figure 12. Behavior of top clients in `mm-image`. Vertical lines in the last-row subfigures indicate average lengths, and error bars show the range of average lengths within a day.



(a) Input and output length distribution, in one-hour windows.



(b) Breakdown correlation. (c) Output length breakdown.

Figure 13. Characterization of input and output lengths for the `deepseek-r1` workload in one day. Error bars in (a) indicate the range of average lengths over the day.

of reasoning models (§2.1) results in longer, more variable output lengths and a distinct ratio of *reason* and *answer* tokens (§5.1). In addition, request arrivals in reasoning workloads are less bursty, partly owing to a considerable proportion of multi-turn conversations, which alter the request arrival pattern (§5.2). We conclude with client decomposition to extend our causal modeling to reasoning workloads (§5.3).

5.1 Understanding Reason & Answer Lengths

Figure 13 characterizes request lengths in the `deepseek-r1` workload, depicting also the reason and answer parts of outputs. In the upper part of Figure 13(a), we observe similar power-law distributions and shifting patterns (Finding 3 and 4) in terms of input and output lengths, as indicated by the fitting curves and error bars. However, output lengths are significantly longer and more variable than those in non-reasoning workloads, due to the long reason lengths. In fact, as shown in the lower part of Figure 13(a), reason lengths can be on average 4× longer than answer lengths, and contribute more to the shifting of output lengths. The different matching levels of the Exponential curves suggest that, to some extent, the reasoning output behaves more like further input for LLMs, while the answer section remains akin to traditional output.

Moreover, Figures 13(b) and 13(c) reveal a non-trivial relation between reason and answer lengths: there exists a clearer correlation between them (compared with Figure 4), while their per-request ratio exhibits a consistent *bimodal* distribution. The bimodality originates from two dominating task patterns adopted by a reasoning model (*i.e.,* reasoning for either a more complete or more concise answer), which future serving optimizations may be able to leverage.

> **Finding 9**: Reasoning workloads exhibit longer and more variable output lengths, due to the reason tokens. In relation, reason and answer lengths display stronger positive correlation, as well as a unique bimodal ratio.

5.2 Arrival and Multi-Turn Patterns

Non-bursty arrivals. Figure 14 illustrates the arrival pattern for both `deepseek-r1` and `deepqwen-r1` over a day. On the left, the CV of request arrivals remains mostly close to or even less than 1 despite the diurnal rate shift, indicating that both workloads are non-bursty (especially compared with those in Figure 2). The right side of Figure 14 further validates this fact, showing that the Exponential distribution fits the inter-arrival time distribution quite well (*i.e.,* the arrival is roughly modeled by Poisson processes).

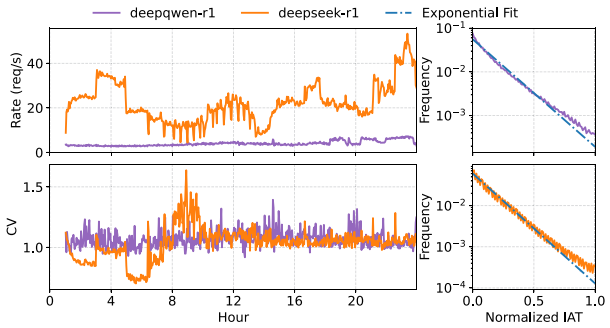
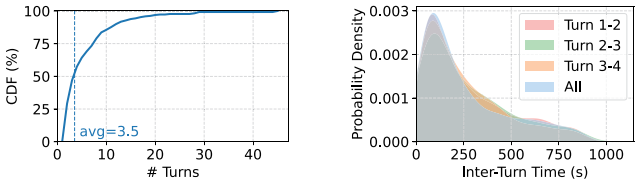


Figure 14. Characterization of request arrival patterns in `deepseek-r1` and `deepqwen-r1`. *Left*: Rate and burstiness shifts over a day. *Right*: Normalized inter-arrival time distributions.



(a) CDF of conversation turns. (b) PDF of inter-turn times.

Figure 15. Characterization of conversations in `deepseek-r1`.

Characterizing multi-turn conversations. Holding multi-turn conversations is an essential capability of LLMs [45, 51], and it also introduces a special pattern to request arrivals: intuitively, earlier requests foretell the *reoccurrence* of follow-up conversations, which may alter the workload burstiness.

We thus conduct a dedicated characterization of multi-turn requests found in `deepseek-r1`. Within our 12-hour analysis window, we have identified[3] 188,986 multi-turn requests out of 1,964,415 total requests, forming 57,205 conversations. Figure 15(a) shows the distribution of the conversation lengths, averaging 3.5. The distribution of inter-turn time (ITT), *i.e.,* the time between the arrival of consecutive turns, is detailed in Figure 15(b). In general, ITTs concentrate around 100 seconds, with an extremely long tail (the figure is truncated at the 75_{th} percentile for visualization).

Impact of multi-turn conversations. Since multi-turn requests constitute almost 10% of the `deepseek-r1` workload, their pattern has a specific impact on workload characteristics. To demonstrate this, we apply two upsampling methods to the identified multi-turn requests, scaling them to the same size as the original workload. The *Naive* method is agnostic about the conversations and simply scales the inter-arrival time, while the *ITT* method works by scaling the arrival time between conversations, leaving the ITT distribution unchanged. Figure 16 compares the upsampled and original workloads by measuring the workload burstiness over time, highlighting a substantial difference: *Naive* produces a highly bursty workload, while the *ITT*-workload is even more stable than the original. It is thus essential for realistic workloads to faithfully

[3]Our method is not accurate for many reasons: parts of conversations could fall out of the analyzed window, or the messages could be altered by the log store. Still, the resulting workload is reasonably large for analysis.

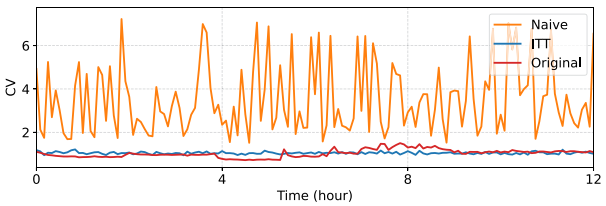
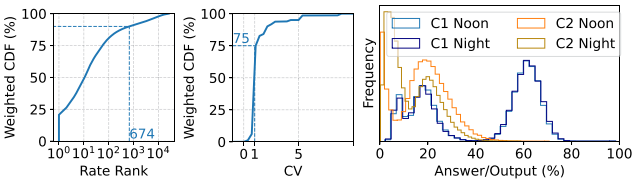


Figure 16. Comparison of two upsampling methods for a workload containing only multi-turn requests.



(a) (b) (c)

Figure 17. Client decomposition for `deepseek-r1`. (a) weighted CDF of client arrival rate. (b) weighted CDF of client burstiness. (c) output length breakdown of top clients (C1 and C2).

reflect the multi-turn conversation pattern by adhering to ITT distributions in Figure 15(b).

> **Finding 10**: Request arrival in reasoning workloads is impacted by the reoccurring pattern of multi-turn conversations and appears less bursty.

5.3 Decomposition of Reasoning Workloads

Figure 17 presents client behaviors in the reasoning workload `deepseek-r1`. Interestingly, according to Figure 17(a), top clients in `deepseek-r1` are shown to be less substantial in comparison with other workloads (Figures 5 and 11): out of 25,913 clients, the top 10 clients only constitute half of the requests. Furthermore, the proportion of non-bursty clients (Figure 17(b)) is also significantly higher, likely contributing to the overall non-burstiness of the workload. In addition, we observe again the bimodal distribution in the breakdown of request output lengths across multiple top clients, as depicted in Figure 17(c). This implies that the pattern revealed in Figure 13(c) can still be causally modeled on a per-client basis, where the day-and-night shift of the answer length ratio is attributed to the fluctuation of client rates.

> **Finding 11**: Clients in reasoning workloads exhibit less skewed rates and less bursty arrivals, while also showing the bimodal pattern in terms of data distributions.

6 Workload Generation

Motivated by the many findings in our characterization, we build ServeGen, a principled framework for generating workloads that incorporate the realistic characteristics revealed in previous sections. Next, we describe our framework (§6.1), validate its accuracy (§6.2), and show its benefits for benchmarking serving systems with a real-world use case (§6.3).

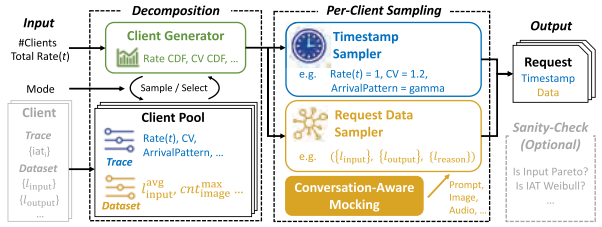


Figure 18. Overview of the ServeGen framework. The color gray indicates optional requirements; *e.g.,* users can still use ServeGen without providing additional client information.

6.1 ServeGen Framework

Figure 18 presents an overview of ServeGen, which is centered around *clients*. Essentially, ServeGen samples requests on a *per-client* basis, and aggregates them to compose realistic workloads. Each client in ServeGen is described by its trace and dataset, both of which can be either parameterized (*e.g.,* modeling a trace with the Gamma distribution) or provided as data samples (*e.g.,* a set of prompt lengths).

To use ServeGen, a user starts by providing the total number of clients, as well as a target total arrival rate. ServeGen then relies on the `Client Generator` to characterize each client, either by sampling from the `Client Pool` preconfigured with realistic client behaviors, or by selecting from a set of user-specified clients with custom traces and datasets. Next, ServeGen samples the request timestamps and data for each client with the `Timestamp Sampler` and `Request Data Sampler`, scaling client rates according to the total rate and generating data via conversation-aware mocking to preserve shared conversation histories. Lastly, ServeGen combines the timestamps and data to produce a final workload.

ServeGen utilizes the findings reported in previous sections to generate realistic workloads and ensure ease of use. For example, given Finding 2, the client rates and the total rate in ServeGen are parameterized over the current time t, enabling workload generation with varying rates. Furthermore, we apply Finding 5 in the `Client Generator` to produce heterogeneous clients (*i.e.,* sampling clients according to real rate and CV) and incorporate other findings on data distributions to configure the `Client Pool` with parameterized real-world clients[4]. Users may optionally use Findings 1 and 3 to sanity-check the overall statistics of generated workloads.

6.2 Generation Accuracy

We validate that the per-client generation approach in ServeGen captures the realistic characteristics of our workloads by measuring the generation accuracy with respect to Findings 2, 4, 6, and 9. Specifically, since the workload arrival patterns and data distributions undergo significant shifts over time, we aim to demonstrate that ServeGen produces workloads that exhibit similar characteristics.

[4]For confidentiality, we release parameterized and sanitized data instead of full data samples. Conversations are selectively released as hashed tokens.

Setups and metrics. We target the variability of data distributions in 3-hour time periods across raw workloads (Actual) and generated workloads that match the overall statistics (*e.g.,* length distributions for the full period). In each period, we measure the average values of relevant request data (*e.g.,* average input lengths for `M-large`) in 3-second windows, and plot them against the request rates in those windows. For language workloads, we explicitly differentiate between stable (*i.e.,* the request rate fluctuates around a certain value) and variable (*i.e.,* the overall request rate is rising or dropping) periods. Intuitively, the shifting patterns in actual workloads should result in visible variability, which ServeGen should be able to match with generated workloads.

Configurations and baselines. We configure ServeGen to select real clients and match the corresponding total rate, effectively resampling the workloads over client decomposition. In contrast, the baseline approach, referred to as NAIVE, directly resamples each workload as a whole to match the overall statistics with ServeGen. NAIVE is meant to represent the workload generation method (*e.g.,* sampling ShareGPT over Poisson processes) used in many existing works [33,49,50,55] when direct workload replay is not an option because published workloads do not exist or mismatch with benchmarking setups. For variable periods, the total rate in NAIVE is also parameterized by time to ensure fair comparisons.

Results. Figure 19 displays the generation accuracy of the two approaches. In every case, the workload produced by ServeGen is shown to be more realistic: the green plot (ServeGen) matches the actual plot much better compared with NAIVE.

Furthermore, the results reveal two major drawbacks of the NAIVE workloads. (i) They can be less variable in terms of request rate, despite their overall burstiness. This is particularly evident during stable periods, where the blue and green scatter plots span considerably wider horizontally, indicating more extreme values for the arrival rate. (ii) They barely capture the correlation between rates and data distributions, which is non-trivial in real workloads (see the blue scatter plot). Such correlations are not surprising given our per-client characterization—large or small short-term rates are likely caused by bursty top clients, and the workload data distributions are expected to shift correspondingly toward or away from the client data distributions.

6.3 Use Case #1: Instance Provisioning

We now put ServeGen to use, illustrating how it helps with benchmarking LLM serving systems by running the generated workloads on vLLM [32], a representative LLM serving system with wide adoption. Particularly, we investigate an *instance-provisioning* scenario, *i.e.,* determining the minimum number of inference instances required to serve a workload while maintaining certain service-level objectives (SLOs). Next, we benchmark a vLLM instance with workloads produced by both ServeGen and the NAIVE approach (as defined

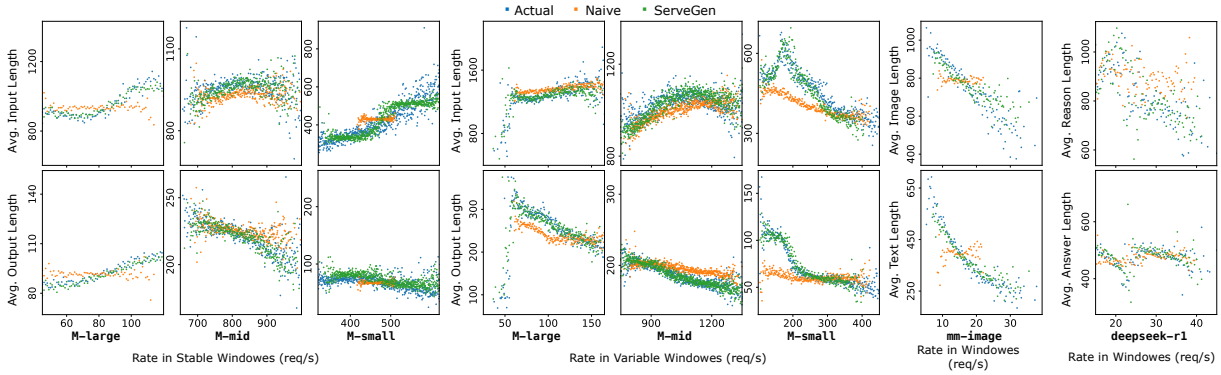
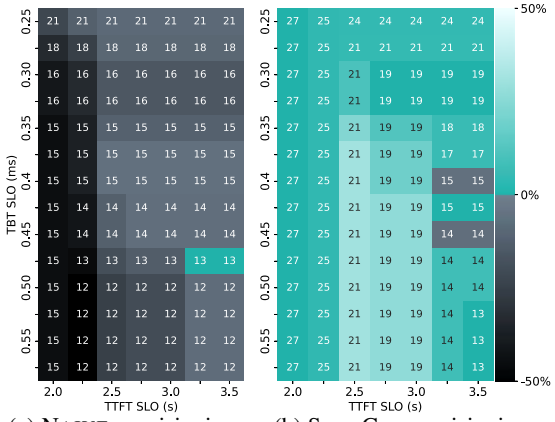


Figure 19. Comparison of workload generation accuracy.



(a) NAIVE provisioning. (b) ServeGen provisioning.

Figure 20. Provisioning results using the NAIVE approach and ServeGen, respectively. In each cell, the number indicates the provisioned instances, while the color shows the over-provisioning percentage.

in §6.2) to obtain provisioning results, and then evaluate how well these results scale when serving real workloads.

Detailed setups. We select a 10-minute period of `M-large` comprising 30,000 requests as the target workload, and set each instance to consist of 2 NVIDIA A100 (80GB) GPUs running a Qwen2.5-14B model[5] with pipeline parallelism [33, 41]. Next, for a grid of target time-to-first-token (TTFT) and time-between-token (TBT) SLOs, we benchmark one instance with workloads generated via both ServeGen and NAIVE, adjusting the workload rate to find the maximum rate each instance can (supposedly) sustain without violating the SLOs (measured as P99 values), and thus derive the number of instances needed in each case. Lastly, we check the results by running the actual `M-large` workload with the provisioned number of instances, recording the actual SLO delivered.

Results. Figure 20 reports the provisioning results, where the number in each heatmap cell represents the provisioned instance count using either NAIVE or ServeGen, and the cell color indicates the over- or under-provisioning percentage. For example, when the target P99 TTFT is 2.25s and TBT is 0.5s, ServeGen results in provisioning 25 instances (4% over the actual number needed), while NAIVE results in only 12 instances (50% under-provisioning). Overall, Figure 20(a) verifies that the NAIVE workloads are *misleadingly easier to serve* than real workloads. Meanwhile, Figure 20(b) fits the actual provisioning results much better, highlighting that the workloads generated by ServeGen can better reflect the system performance in real-world deployment.

6.4 Use Case #2: PD-Disaggregation

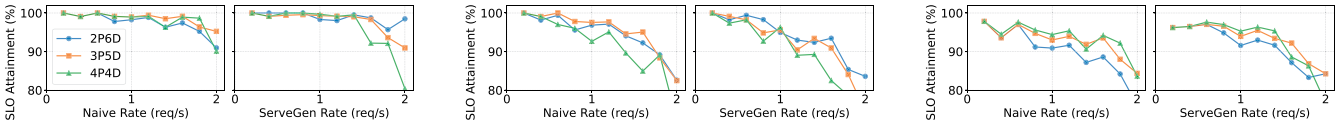
We also demonstrate how ServeGen is beneficial to understanding the impact of serving system design choices with a case study on PD-disaggregation [57], an architectural innovation that is widely adopted in both academia [23, 26, 30] and industry [7, 8, 10, 37]. While it disaggregates the prefill and decoding phases to avoid interference and achieve tailored optimization, recent research [36, 52] indicate that it can be tricky to fully characterize its performance impact in real-world deployments. Here, we show with ServeGen that the lack of proper workload understanding is largely at fault.

Detailed setups. We follow the same steps in §6.3 to generate workloads via ServeGen and NAIVE (with identical overall rate and data distribution). Each workload is then used to benchmark a Qwen2.5-72B model deployed on 4 nodes with 8 H20 GPUs each, running PD-disaggregated SGLang [56] (tensor parallelism [41] set to 4). We vary the prefill and decoding resource scaling (*e.g.,* 3P5D indicates 3 prefill instances and 5 decoding instances) to examine the best configuration.

Results. Figure 21 shows the benchmarking results under different target SLO setups, where PD-disaggregation displays highly workload-sensitive performance. Most interestingly, the results may disagree about the best PD configuration under NAIVE and ServeGen workloads even under similar distributions. In two of the three cases (Figures 21(a) and 21(b)), the best configuration under ServeGen is 2P6D, which happens to be a worse configuration reported by NAIVE benchmarking. This discrepancy is likely due to more intricate patterns of long-tailed requests in ServeGen, leading to more extreme load-imbalance and a greater demand for decoding instances.

In practice, Figure 21 may manifest itself as "unpredictable" performance drops where PD-disaggregation deliv-

[5]We opt for a smaller model than `M-large` due to budget constraints.



(a) Base SLO (b) Tight TBT SLO (c) Tight TTFT SLO

Figure 21. SLO attainment under various setups. **Base** SLO: 8s TTFT, 60ms TBT. **Tight** SLO: 4s TTFT, 30ms TBT (half of **Base**).

ers suboptimal performance due to transient load-imbalance as workloads shift (characterized by ServeGen instead of NAIVE), highlighting the need for new design explorations (*e.g.,* dynamic load-balancing, explicit performance modeling) to further improve the PD-disaggregation approach.

7 Discussion

Fostering future research. The aforementioned findings have already benefited several development teams at Alibaba Cloud Model Studio, including those focused on inference engine optimization, resource planning, and request scheduling. Meanwhile, ServeGen can guide further research on LLM serving in many other ways, and we point out two potential directions here. First, our multimodal workload analysis reveals that a significant portion of TTFT stems from preprocessing (*i.e.,* downloading, normalization, and encoding). This highlights the importance of conducting full-stack optimizations (*e.g.,* decoupling the modality encoders and scaling them independently according to Finding 6, rather than focusing solely on the LLM). Second, our analysis of multi-turn conversations in reasoning workloads reveals that the arrival pattern for these requests is non-bursty (Finding 10), providing valuable insights for improving short-term workload predictability in conversational scenarios.

Limitations of ServeGen. While ServeGen covers mainstream LLM serving workloads, there are several aspects that require further study. First, some complex LLM applications adopt *plugin calls*, where a series of functions are called prior to model inference, performing web searches, database queries, or calling external APIs. The dependent execution of different functions collectively determines the end-to-end execution time, and the output length is influenced as well. We leave characterizing LLM serving with plugin calls as an important area for future work. Second, *prefix caching* [56] enables sharing intermediate KV cache between requests with common prompt prefixes. However, characterizing prefix caching requires access to the content of requests which we currently opt out of due to confidentiality obligations. Third, our analysis of individual workloads does not yet span long enough for *longitudinal* studies (*e.g.,* trends in serving workloads over months).

8 Related Work

LLM serving analysis. Prior work has analyzed many aspects of LLM serving to faciliate optimization and development. Specific to LLM serving *workloads*, BurstGPT [46] and LMM [38] have characterized language and image-to-text model serving workloads, while a series of other studies have performed brief analyses from certain viewpoints such as burstiness [22], computational load [23, 57], prefix-sharing [16, 37], and energy efficiency [42]. On the other hand, dedicated simulators (*e.g.,* AstraSim [48], Vidur [11]) have been built to tackle high-fidelity simulation of serving *systems* for performance analysis. ServeGen serves as a comprehensive characterization of LLM serving workloads with a larger scale and scope, while also strictly complementing inference simulation and hopefully unlocking realistic end-to-end performance modeling.

Workload modeling and generation. Existing research has characterized various workloads in alternative machine learning scenarios such as GPU deep learning [27, 29, 44, 47] and LLM development [28], providing many valuable insights. Some works [13, 15, 18, 31, 47] have also proposed generating realistic workloads by modeling them with historical data, yet they mostly target generic cloud-computing workloads. BurstGPT [46] is a recent work that tackles workload modeling for LLM serving, which uses a parameterized Gamma process to account for variant burstiness in LLM serving. Meanwhile, a large body of prior research [33, 49, 50, 55] on LLM serving has relied on the NAIVE approach and simply combined traces and datasets. We hope the release of ServeGen can foster LLM serving research by covering multiple workload categories and modeling them more accurately with client decomposition, while ensuring ease of use for practitioners.

9 Conclusion

We present a comprehensive characterization of real-world LLM serving workloads for language, multimodal, and reasoning models. We unveil various characteristics and summarize meaningful findings. Based on these findings, we provide ServeGen, a principled framework that generates realistic serving workloads by composing them on a per-client basis. We demonstrate the benefits of ServeGen via case studies on instance provisioning and PD disaggregation.

Acknowledgments. We thank our shepherd, Ramachandran Ramjee, and the anonymous reviewers for their insightful feedback. This work was supported in part by the Scientific Research Innovation Capability Support Project for Young Faculty under Grant ZYGXQNJSKYCXNLZCXM-I1. Xin Jin and Xue Li are the corresponding authors. Yuxing Xiang, and Xin Jin are also with the Key Laboratory of High Confidence Software Technologies (Peking University), Ministry of Education.

References

[1] Disaggregated prefilling and kv cache transfer roadmap in vLLM. https://github.com/vllm-project/vllm/issues/10818, 2024.

[2] Introducing OpenAI o1. https://openai.com/o1/, 2024.

[3] Learning to reason with LLMs. https://openai.com/index/learning-to-reason-with-llms/, 2024.

[4] OpenAI's GPT-4o model. https://openai.com/index/hello-gpt-4o/, 2024.

[5] Qwen2.5: A party of foundation models. https://qwenlm.github.io/blog/qwen2.5/, 2024.

[6] Sharegpt-52k. https://huggingface.co/datasets/RyokoAI/ShareGPT52K, 2024.

[7] DeepSeek-V3/R1 inference system overview. https://github.com/deepseek-ai/open-infra-index/blob/main/202502OpenSourceWeek/day_6_one_more_thing_deepseekV3R1_inference_system_overview.md, 2025.

[8] Deploying DeepSeek with PD disaggregation and large-scale expert parallelism on 96 H100 GPUs. https://lmsys.org/blog/2025-05-05-large-scale-ep/, 2025.

[9] Grok 3 beta — the age of reasoning agents. https://x.ai/blog/grok-3/, 2025.

[10] NVIDIA Dynamo. https://github.com/ai-dynamo/dynamo, 2025.

[11] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee, and Alexey Tumanov. VIDUR: A large-scale simulation framework for LLM inference. In *Proceedings of Machine Learning and Systems*, 2024.

[12] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. In *arXiv*, 2024.

[13] Arshdeep Bahga and Vijay Krishna Madisetti. Synthetic workload generation for cloud computing applications. In *Journal of Software Engineering and Applications*, 2011.

[14] Luiz Andr Barroso, Jimmy Clidaras, and Urs Hlzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool Publishers, 2nd edition, 2013.

[15] Shane Bergsma, Timothy Zeyl, Arik Senderovich, and J. Christopher Beck. Generating complex, realistic cloud workloads using recurrent neural networks. In *ACM SOSP*, 2021.

[16] Shiyi Cao, Yichuan Wang, Ziming Mao, Pin-Lun Hsu, Liangsheng Yin, Tian Xia, Dacheng Li, Shu Liu, Yineng Zhang, Yang Zhou, Ying Sheng, Joseph Gonzalez, and Ion Stoica. Locality-aware fair scheduling in LLM serving. In *arXiv*, 2025.

[17] Xiaokang Chen, Zhiyu Wu, Xingchao Liu, Zizheng Pan, Wen Liu, Zhenda Xie, Xingkai Yu, and Chong Ruan. Janus-Pro: Unified multimodal understanding and generation with data and model scaling. In *arXiv*, 2025.

[18] Eli Cortez, Anand Bonde, Alexandre Muzio, Mark Russinovich, Marcus Fontoura, and Ricardo Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *ACM SOSP*, 2017.

[19] DeepSeek-AI. DeepSeek-V3 technical report. In *arXiv*, 2024.

[20] DeepSeek-AI. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. In *arXiv*, 2025.

[21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[22] Jiangfei Duan, Runyu Lu, Haojie Duanmu, Xiuhong Li, Xingcheng Zhang, Dahua Lin, Ion Stoica, and Hao Zhang. MuxServe: flexible spatial-temporal multiplexing for multiple LLM serving. In *International Conference on Machine Learning (ICML)*, 2024.

[23] Pratyush Patel et al. Splitwise: Efficient generative LLM inference using phase splitting. In *arXiv*, 2024.

[24] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. ServerlessLLM: Low-latency serverless inference for large language models. In *USENIX OSDI*, 2024.

[25] Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. DeepSpeed-FastGen: High-throughput text generation for LLMs via MII and DeepSpeed-Inference. In *arXiv*, 2024.

[26] Ke Hong, Lufang Chen, Zhong Wang, Xiuhong Li, Qiuli Mao, Jianping Ma, Chao Xiong, Guanyu Wu, Buhe Han, Guohao Dai, Yun Liang, and Yu Wang. semi-PD: Towards efficient LLM serving via phase-wise disaggregated computation and unified storage. In *arXiv*, 2025.

[27] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. Characterization and prediction of deep learning workloads in large-scale GPU datacenters. In *ACM SC*, 2021.

[28] Qinghao Hu, Zhisheng Ye, Zerui Wang, Guoteng Wang, Meng Zhang, Qiaoling Chen, Peng Sun, Dahua Lin, Xiaolin Wang, Yingwei Luo, Yonggang Wen, and Tianwei Zhang. Characterization of large language model development in the datacenter. In *USENIX NSDI*, 2024.

[29] Myeongjae Jeon, Shivaram Venkataraman, Amar Phanishayee, Junjie Qian, Wencong Xiao, and Fan Yang. Analysis of large-scale multi-tenant GPU clusters for DNN training workloads. In *USENIX ATC*, 2019.

[30] Yibo Jin, Tao Wang, Huimin Lin, Mingyang Song, Peiyang Li, Yipeng Ma, Yicheng Shan, Zhengfan Yuan, Cailong Li, Yajing Sun, Tiandeng Wu, Xing Chu, Ruizhi Huan, Li Ma, Xiao You, Wenting Zhou, Yunpeng Ye, Wen Liu, Xiangkun Xu, Yongsheng Zhang, Tiantian Dong, Jiawei Zhu, Zhe Wang, Xijian Ju, Jianxun Song, Haoliang Cheng, Xiaojing Li, Jiandong Ding, Hefei Guo, and Zhengyong Zhang. P/D-Serve: Serving disaggregated large language model at scale. In *arXiv*, 2024.

[31] Da-Cheng Juan, Lei Li, Huan-Kai Peng, Diana Marculescu, and Christos Faloutsos. Beyond poisson: Modeling inter-arrival time of requests in a datacenter. In *PAKDD*, 2014.

[32] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *ACM SOSP*, 2023.

[33] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, Vincent Liu, Ying Sheng, Xin Jin, Yanping Huang, Zhifeng Chen, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *USENIX OSDI*, 2023.

[34] Jiachen Liu, Zhiyu Wu, Jae-Won Chung, Fan Lai, Myungjin Lee, and Mosharaf Chowdhury. Andes: Defining and enhancing quality-of-experience in LLM-based text streaming services. In *arXiv*, 2024.

[35] Chengzhi Lu, Kejiang Ye, Guoyao Xu, Cheng-Zhong Xu, and Tongxin Bai. Imbalance in the cloud: An analysis on Alibaba cluster trace. In *International Conference on Big Data*, 2017.

[36] Tiyasa Mitra, Ritika Borkar, Nidhi Bhatia, Ramon Matas, Shivam Raj, Dheevatsa Mudigere, Ritchie Zhao, Maximilian Golub, Arpan Dutta, Sailaja Madduri, Dharmesh Jani, Brian Pharris, and Bita Darvish Rouhani. Beyond the buzz: A pragmatic take on inference disaggregation. In *arXiv*, 2025.

[37] Ruoyu Qin, Zheming Li, Weiran He, Jialei Cui, Feng Ren, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. Mooncake: Trading more storage for less computation — a KVCache-centric architecture for serving LLM chatbot. In *USENIX FAST*, 2025.

[38] Haoran Qiu, Anish Biswas, Zihan Zhao, Jayashree Mohan, Alind Khare, Esha Choukse, Íñigo Goiri, Zeyu Zhang, Haiying Shen, Chetan Bansal, Ramachandran Ramjee, and Rodrigo Fonseca. Towards efficient large multimodal model serving. In *arXiv*, 2025.

[39] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *ACM Symposium on Cloud Computing*, 2012.

[40] Mohammad Shahrad, Rodrigo Fonseca, Inigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark Russinovich, and Ricardo Bianchini. Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider. In *USENIX ATC*, 2020.

[41] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. In *arXiv*, 2020.

[42] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. In *arXiv*, 2024.

[43] Abhishek Verma et al. Evaluating job packing in warehouse-scale computing. In *International Conference on Cluster Computing*, 2014.

[44] Mengdi Wang, Chen Meng, Guoping Long, Chuan Wu, Jun Yang, Wei Lin, and Yangqing Jia. Characterizing deep learning training workloads on Alibaba-PAI. In *International Symposium on Workload Characterization (IISWC)*, 2019.

[45] Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *arXiv*, 2024.

[46] Yuxin Wang, Yuhan Chen, Zeyu Li, Xueze Kang, Yuchu Fang, Yeju Zhou, Yang Zheng, Zhenheng Tang, Xin He, Rui Guo, Xin Wang, Qiang Wang, Amelie Chi Zhou, and Xiaowen Chu. BurstGPT: A real-world workload dataset to optimize LLM serving systems. In *ACM SIGKDD*, 2025.

[47] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters. In *USENIX NSDI*, 2022.

[48] William Won, Taekyung Heo, Saeed Rashidi, Srinivas Sridharan, Sudarshan Srinivasan, and Tushar Krishna. ASTRA-sim2.0: Modeling hierarchical networks and disaggregated systems for large-model training at scale. In *International Symposium on Performance Analysis of Systems and Software*, 2023.

[49] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. LoongServe: Efficiently serving long-context large language models with elastic sequence parallelism. *ACM SOSP*, 2024.

[50] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models. In *arXiv*, 2024.

[51] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *arXiv*, 2023.

[52] Yu Wu, Tongxuan Liu, Yuting Zeng, Siyu Wu, Jun Xiong, Xianzhe Dong, Hailong Yang, Ke Zhang, and Jing Li. Arrow: Adaptive scheduling mechanisms for disaggregated LLM inference architecture. In *arXiv*, 2025.

[53] Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, Bin Zhang, Xiong Wang, Yunfei Chu, and Junyang Lin. Qwen2.5-Omni technical report. In *arXiv*, 2025.

[54] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for transformer-based generative models. In *USENIX OSDI*, 2022.

[55] Hong Zhang, Yupeng Tang, Anurag Khandelwal, and Ion Stoica. SHEPHERD: Serving DNNs in the wild. In *USENIX NSDI*, 2023.

[56] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *arXiv*, 2024.

[57] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *USENIX OSDI*, 2024.