ElasticMM: Efficient Multimodal LLMs Serving with Elastic Multimodal Parallelism

Zedong Liu[1,2], **Shenggan Cheng**[3], **Guangming Tan**[1], **†Yang You**[3], and **†Dingwen Tao**[1]

[1]Institute of Computing Technology, Chinese Academy of Sciences
[2]University of Electronic Science and Technology of China
[3]National University of Singapore

Abstract

Multimodal large language models (MLLMs) extend LLMs to handle images, videos, and audio by incorporating feature extractors and projection modules. However, these additional components—combined with complex inference pipelines and heterogeneous workloads—introduce significant inference overhead. Therefore, efficiently serving MLLMs remains a major challenge. Current tightly coupled serving architectures struggle to distinguish between mixed request types or adapt parallelism strategies to different inference stages, leading to increased time-to-first-token (TTFT) and poor resource utilization. To address this, we introduce Elastic Multimodal Parallelism (EMP), a new serving paradigm that elastically adapts to resource heterogeneity across request types and inference stages. Building upon EMP, we develop ElasticMM, an MLLM serving system that (1) separates requests into independent modality groups with dynamic resource allocation via a modality-aware load balancer; (2) decouples inference stages and enables parallelism adjustment and adaptive scaling via elastic partition scheduling; and (3) improves inference efficiency through unified multimodal prefix caching and non-blocking encoding. Experiments on diverse real-world datasets show that ElasticMM outperforms state-of-the-art (SOTA) serving systems, reducing TTFT by up to 4.2× and achieving 3.2–4.5× higher throughput while meeting service-level objectives (SLOs).

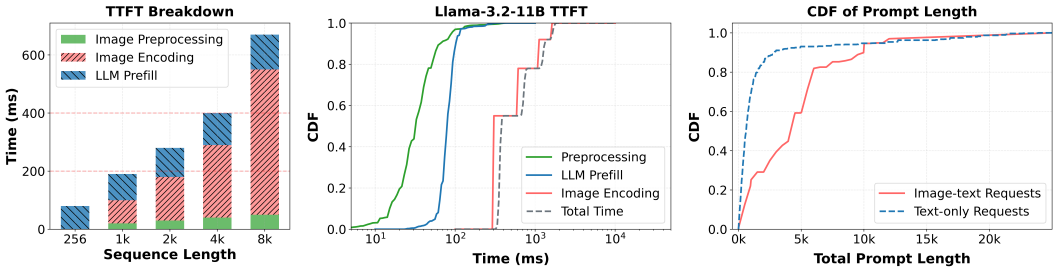
1 Introduction

Large Language Models (LLMs) have had a profound impact on real-world applications due to their exceptional performance [1–3]. Their gradual expansion into the multimodal domain has led to the emergence of Multimodal Large Language Models (MLLMs), which can process inputs such as images, videos, and audio [4, 5]. By employing feature extractors and projection modules to map multimodal inputs into the LLM feature space, MLLMs excel at tasks like visual question answering (VQA), image captioning, and multimodal interaction [6–9]. Consequently, MLLMs are increasingly deployed in online services, where they are highly sensitive to service-level objectives (SLOs).

Unlike traditional LLMs that process only text [10–14], MLLMs must handle diverse input types, introducing additional model components and a more complex inference pipeline. This pipeline typically includes distinct stages such as image preprocessing, image encoding, and language model inference (comprising prefill and decode). These added stages increase computational complexity (Fig. 1b) and significantly extend the average context length (Fig. 1c). However, existing inference serving systems adopt a tightly coupled architecture that neither distinguishes between request types

[0]† Corresponding Authors.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).



(a) TTFT breakdown **(b)** CDF of inference stage latencies **(c)** CDF of prompt lengths

Figure 1: MLLMs' inference overhead and workload complexity. (a) and (b) demonstrate the significant overhead introduced by MLLMs. (c) reveals the longer context in multimodal requests. Results obtained using the LLaMA3.2-11B model on the ShareGPT-4o dataset.

nor decouples inference components, executing all stages on the same hardware instances [15–18]. As a result, time-to-first-token (TTFT) *increases sharply* under heavy multimodal workloads. Furthermore, encoder-decoder models introduce cross-attention layers [19–21], and the computational heterogeneity between text-only and multimodal requests *reduce batching efficiency*.

To address this issue, a decoupled multimodal inference architecture emerges as a promising solution—processing text-only and multimodal requests on separate instances. Our systematic analysis reveals that multimodal workloads exhibit bursty patterns, often characterized by sudden spikes in image inputs (as also observed in previous work [22]), and that different inference stages benefit from varying degrees of parallelism. For example, image encoding and LLM prefill stages have higher computational complexity and benefit from large-scale parallelism, whereas the decode stage has limited scalability [23–25]. These observations motivate two critical design challenges: (1) static resource allocation struggles to adapt to dynamically changing workloads, and (2) fixed parallelism strategies often fail to match the varying resource demands of different inference stages.

To this end, we propose **Elastic Multimodal Parallelism (EMP)**. To address the first challenge, we introduce a *modality-aware load balancing technique* that monitors real-time workload fluctuations and dynamically adjusts resource allocation across different modality groups. To tackle the second challenge, we design an *elastic partitioned scheduling technique*, which enables dynamic parallelism adaptation at the granularity of each inference stage. For example, compute-intensive stages such as encoding and prefill are scaled across more GPUs, while the decode stage reduces parallelism to free up resources for other requests.

Building on EMP, we present **ElasticMM**, an MLLM serving system with four key contributions:

- We identify tight coupling in existing systems as a major bottleneck for MLLM serving and propose EMP, a new serving paradigm that separates text-only and multimodal requests into independent modality groups and decouples inference stages for stage-specific parallelism.
- We design two core techniques for EMP: (1) ***modality-aware load balancing***, which dynamically allocates and scales resources across modalities to handle unpredictable workloads; and (2) ***elastic partitioned scheduling***, which controls parallelism through flexible scheduling and scaling to maximize inference performance.
- We propose two optimizations to mitigate MLLM-specific overheads: ***unified multimodal prefix caching*** to reduce redundant computation and data transfer, and ***non-blocking encoding*** to minimize the impact of encoding latency on the overall inference pipeline.
- We conduct a comprehensive evaluation of ElasticMM on two real-world datasets. Compared to the SOTA baseline vLLM [16], ElasticMM reduces TTFT by up to 4.2× and achieves a 3.2×–4.5× throughput improvement while meeting service-level objective (SLO) requirements.

2 Background and Motivation

2.1 Multimodal Large Language Models Inference

MLLM Inference Pipeline. MLLMs extend traditional language models by integrating visual, auditory, and other modalities, enabling unified reasoning across heterogeneous input sources for

Table 1: Model configurations for four representative MLLMs with input image of 904×904 pixels.

MLLM Model Name	Architecture	Image Encoder *(#Params)*	Total Image Token Size	LLM Backend *(#Params)*
Llama3.2-Vision 11B	Encoder-Decoder	ViT-H/14 (630M)	6516	Llama 3.1 (8B)
Llama3.2-Vision 90B	Encoder-Decoder	ViT-H/14 (630M)	6516	Llama 3.1 (70B)
Qwen2.5-VL 7B	Decoder-only	ViT (670M)	7410	Qwen2.5 (7B)
Qwen2.5-VL 72B	Decoder-only	ViT (670M)	7410	Qwen2.5 (72B)

complex multimodal tasks [1–3, 5]. Taking vision-language models as an example, the MLLM inference pipeline typically consists of three key stages: (1) Image Preprocessing: Raw images are resized and divided into uniformly sized tiles [26]. (2) Image Encoding: A vision encoder extracts visual features and converts them into vision tokens [27, 28]. (3) Text Generation: A language model takes both vision tokens and a text prompt as input to generate responses.

MLLM Architectures. Modern MLLMs generally fall into two architectural categories: (1) Decoder-only (DecOnly) architectures, such as LLaVA-OneVision [8], Qwen-VL [29], DeepSeek's Janus [30], and InternVL [7]; and (2) Encoder-decoder (EncDec) architectures, including LLaMA-3.2 Vision [21], NVLM-X [20], and Flamingo [19]. The key difference lies in how vision and text tokens are processed. Decoder-only models concatenate vision and text tokens and feed them together into the language model; vision tokens participate in every generation step. In contrast, encoder-decoder models use cross-attention modules to align multimodal inputs: vision tokens interact with text tokens only via these cross-attention layers, which are inserted between self-attention layers.

Additional Overheads. With the continuous increase in model scale and complexity, the computational cost of inference also grows, especially for MLLMs. Compared to language models, MLLMs introduce overhead from two main sources (shown in Fig. 1a): (1) Increased Architectural Complexity: extra components such as vision encoders and cross-attention layers make the model heavier. (2) Extended Context Length: Multimodal data, once encoded into tokens, are concatenated with text prompts, increasing input length during inference [26]. Specifically, this extension increases computational load, memory usage, and causes degradation in latency and throughput. Table 1 illustrates the encoder sizes and vision token lengths added by four mainstream open-source MLLMs.

2.2 LLM Serving Systems

Existing LLM Serving. Existing serving systems [31, 17, 32], such as vLLM [16] and SGLang [18], have introduced a range of techniques to accelerate inference. To avoid redundant computation, these systems cache the key-value (KV) states of tokens for reuse. This design splits the inference process into two stages: the prefill stage and the decode stage. The prefill stage computes the KV cache for all input tokens and generates the first output token, while the decode stage generates one token per iteration. Consequently, the prefill stage carries a significantly heavier computational load. To this end, ORCA [33] introduced continuous batching, enabling the system to process the prefill and decode stages of multiple requests in an uninterrupted stream of batches. To alleviate the burden of long input contexts, chunked prefill [25, 34] splits long contexts into smaller segments that can be interleaved with decoding. However, interference between the two stages remains difficult to eliminate. A promising solution is prefill-decoding disaggregation, which places the two stages on separate GPU instances [23, 24, 35]. LoongServe [36] further introduces elastic sequence parallelism to further optimize parallelism and resource allocation under this disaggregated architecture.

Coupled Multimodal Serving. Despite these advancements, SOTA serving systems exhibit significant coupling issues when deployed for MLLMs. This coupling occurs at both the service and infrastructure levels. At the service level, existing systems treat unimodal (text-only) and multimodal requests identically, routing both through the same inference pipeline, despite their vastly different computational requirements. At the infrastructure level, all components—including preprocessors, vision encoders, and LLM backends—are colocated on the same hardware server. These tightly coupled components must share compute and memory resources while being constrained to a uniform batching and model parallelism strategy, leading to resource contention and inefficiency.

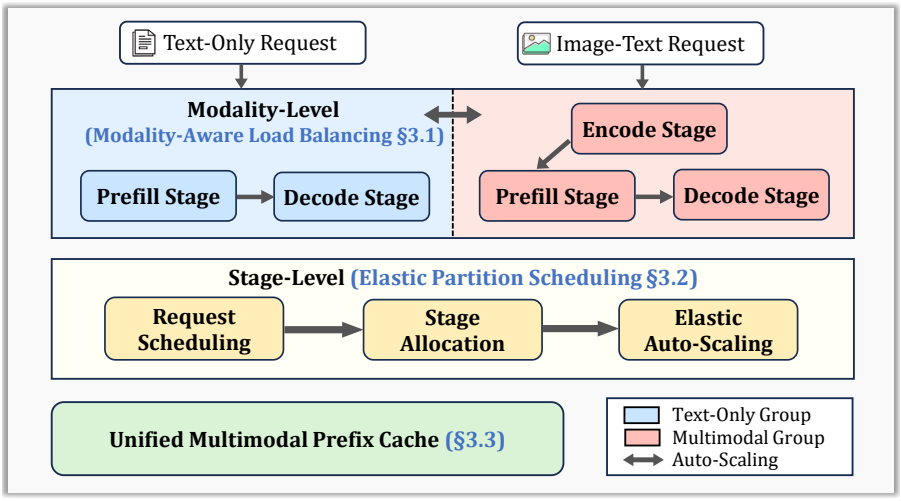


Figure 2: Framework diagram of ElasticMM. The figure illustrates a two-level scheduling framework that collaboratively enables elastic multimodal parallelism.

2.3 Research Challenges and Motivations

Existing tightly coupled systems face significant limitations when serving multimodal models. ***Service-level Problem:*** Multimodal and text-only requests differ substantially in resource demands due to longer context lengths and the injection of multimodal data (as illustrated in Fig. 1c). Meeting the same SLO requires different resource allocations for each request type. However, coupling their inference execution reduces overall resource utilization and increases the risk of SLO violations. It also prevents leveraging their distinct characteristics to enable more efficient inference and resource scheduling. ***Architectural Problem:*** Encoder-decoder models[19–21], which incorporate cross-attention mechanisms, are ill-suited for mixed-batch processing. Combining both request types in a single batch not only increases the latency of text-only requests but also degrades overall throughput.

These lead to our ***Key Insight 1:*** To better meet the distinct requirements of each request type, a *modality-aware decoupled* inference architecture should be adopted, where text-only and multimodal requests are processed independently.

However, naive static decoupling with fixed resource allocation is ineffective under dynamic workloads. It cannot adapt to changing request distributions (e.g., sudden spikes in image traffic) or adjust parallelism strategies based on evolving resource demands.

This leads to our ***Key Insight 2:*** An *elastic* serving system is essential for handling dynamic multimodal workloads. Such a system must extend static architectures to support dynamic resource reallocation and stage-specific parallelism adjustments. By enabling scalable execution across elastic instances, it effectively alleviates compute and memory bottlenecks in multimodal serving.

3 Elastic Multimodal Parallelism

As discussed in the previous section, it is essential to build an elastic MLLM serving system that can dynamically adjust both resource allocation and parallelism strategies. To this end, we introduce a hierarchical framework that enables elastic scheduling, where all instances are organized into two levels, as illustrated in Fig. 2.

At the first level, the **modality level**, instances are grouped based on the modality of the models they serve (e.g., text-only or multimodal). At the second level, the **stage level**, the inference pipeline is further disaggregated into distinct stages, such as encoding, prefill, and decode. At each level, our system provides both ***decoupling*** and ***elasticity***, which are the two core strengths of ElasticMM. This design eliminates resource contention and maximizes utilization.

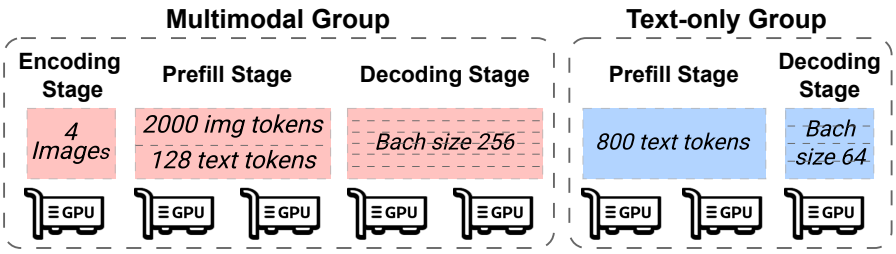


Figure 3: Illustration of the elastic scheduling space in EMP.

At the modality level, we design a *modality-aware load balancing strategy* to dynamically allocate resources across modality groups in response to fluctuating request loads. At the stage level, we apply *elastic partition scheduling* to enable flexible parallelism tailored to the needs of each inference stage.

3.1 Modality-Aware Load Balancing

To tackle the inter-modality load imbalance outlined in Section 2.3, we combine both proactive and reactive mechanisms. Specifically, following prior approaches [37–39], we apply a proactive mechanism to allocate resources to different modality groups. However, proactive mechanism alone cannot effectively handle sudden request surges (e.g., bursty multi-image streams). We therefore design a reactive scaling mechanism to dynamically expand capacity upon detecting resource shortages.

Proactive Mechanism. Our analysis of load patterns in inference services reveals twofold: although the short-term arrival pattern of a single request stream is difficult to predict, aggregating multiple streams results in smooth and periodic patterns (e.g., lower load at night, higher during the day). As observed in previous work [22], text-only requests exhibit stable loads, while multimodal traffic is marked by pronounced surges. Based on the predictability of long-term workload, idle elastic instances can be proactively assigned to modality groups. Following previous work [37], our goal in allocation is to maximize the minimum burst tolerance across modality groups. Burst tolerance (bt) is defined as the ratio between peak-available and average-required instances per group in Equation 1. To achieve this, we employ a fast and effective greedy strategy: incrementally assigns each instance to the group currently lowest burst tolerance, continuing until resources are fully allocated.

$$bt(i) = \frac{\text{\# Instances } i \text{ can use for its peak load}}{\text{\# Instances } i \text{ can use for its average load}} = \frac{N_i^{\text{peak}}}{N_i^{\text{avg}}} \tag{1}$$

Reactive Scaling. Due to unpredictable short-term workloads, the system may face sudden traffic bursts, such as long-text or image-heavy requests in real-time scenarios. The system evaluates the trade-off between adjusting intra-group parallelism and triggering inter-group reactive scaling, based on a gain-cost model described in Section 3.2. If inter-group reactive scaling is more beneficial, the modality-level manager selects instances to preempt from other groups with minimal impact. When an instance E is preempted, its workload is merged into other instances at the same inference stage.

3.2 Elastic Partition Scheduling

After addressing inter-group GPU allocation, we turn to intra-group request scheduling and parallelism adjustment, adapting to the distinct characteristics of each inference stage. For example, prefill is compute-bound; decode is always memory-bound and scales poorly [23, 24, 35]. For stages with sublinear scalability, allocating more idle GPUs improves performance. Conversely, for stages with poor scalability, it is more efficient to limit the number of GPUs involved [36]. Fig. 3 illustrates request assignments and parallel execution modes in the ElasticMM service, showing that instance counts vary by modality group and inference stage. We propose Elastic Partition Scheduling, orchestrated by the stage-level manager. This strategy decomposes the scheduling challenge into three subproblems: (1) ***Request scheduling***, (2) ***Stage Allocation***, and (3) ***Elastic auto-scaling***.

Request Dispatching. This step selects a subset of pending requests $R_p \subseteq P$ from the queue P a First-Come-First-Served (FCFS) policy for the prefill phase [16, 33]. One exception: if a text-only dialogue is redirected to a multimodal group due to associated multimodal requests, it's prioritized—this helps

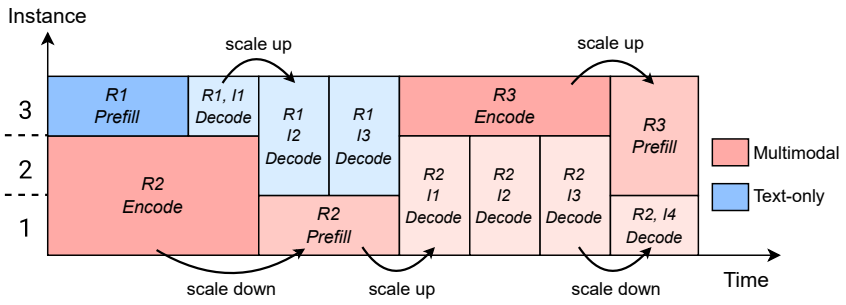


Figure 4: Example of elastic auto-scaling in three instance.

overlap migration overhead and frees KV slots earlier. The dispatch process must address two types of constraints: GPU memory and compute throughput. For memory constraints, the manager only adds requests to R_p if sufficient KV slots are available [40]. There's a tipping point where the system shifts from memory-bound to compute-bound. Before this point, adding requests to R_p improves utilization; after that, additional requests degrade performance due to extended execution time. We estimate this point by analyzing the upper bound of prefill time under memory bound.

Stage Allocation. Once the prefill request set R_p is formed, the system allocates a set of elastic instances E_p to serve it, aiming to maximize GPU efficiency. Inspired by Loongserve [36], the system will prioritize allocating idle instances to R_p for prefill stage. If the available KV cache slots in idle instances are insufficient, R_p is permitted to preempt instances set E_d in the decoding stage B_d. The instance with maximum unused slots are denoted as e_{max}. Meanwhile, drawing insights from existing research, compute-intensive stages like prefill can still benefit from increased parallelism. Therefore, the stage-level manager continuously considers further preemptions by selecting the e_{max}, migrating its KV cache to other decode instances. The gain-cost model is formulated as follows:

$$\text{Gain} = \sum_{r \in R_p} \frac{T(R_p, E_p) - T(R_p, E_p \cup e_{max})}{r.\text{input\_len}} \quad \text{Cost} = \sum_{r \in B_d} \frac{M(e_{max}) + w \cdot L(B_d, E_d - e_{max})}{r.\text{output\_len}} \tag{2}$$

The gain of preemption is quantified as the acceleration gained by adding e_{max} to the prefill process. The cost accounts for both migration overhead $M(e_{max})$ and the performance impact L on the preempted computation. A tunable penalty factor w is introduced to control the aggressiveness of preemption, enabling flexible system behavior under varying workloads. Within a single inference stage, we prioritize Data Parallelism (DP). Alternatives such as Tensor Parallelism are only employed when a single GPU cannot hold the model weights. This design offers several advantages: 1) During elastic scaling, only the KV cache needs to be migrated, avoiding expensive weight transfers; 2) DP enables more flexible utilization of fragmented GPU resources, whereas tensor parallelism typically requires an even number of devices and stricter placement constraints.

Elasic Auto-Scaling. ElasticMM supports elastic scaling, including at the stage level. Since resource allocation is primarily guided by prefill, we monitor the decode phase to trigger elastic adjustments. Given that the decode stage exhibits poor scalability, we shrink it to the minimum parallelism. When the GPU resource becomes insufficient during decoding, the system triggers reactive scaling. Based on prior observations, decode bottlenecks often occur at FFN layers and depend on batch size. We establish scaling thresholds through offline profiling. The scaling-up first tries to allocate idle instances from the same group. If that fails, it first selects a candidate instance e_{max} from intra-group prefill instances and a candidate e'_{max} from inter-group instances, using the following gain-cost model to estimate the trade-off between potential speedup and migration overhead. The instance with the highest net gain is selected for preemption. If inter-group preemption offers greater benefit, the system triggers the reactive scaling mechanism described in Section 3.1.

$$\text{Gain} = \sum_{r \in B_d} \frac{\text{AvgLat}_d - T(B_d, E_d \cup e_{max})}{r.\text{output\_len}} \quad \text{Cost} = \sum_{r \in R'_p} \frac{M(e_{max}) + w \cdot L(R'_p, E'_p - e_{max})}{r.\text{input\_len}} \tag{3}$$

Fig. 4 visualizes auto-scaling. Given the higher computational complexity of multimodal encoding compared to text-only prefill and decode, $R2$'s encode stage is initially allocated more resources. During $R1$'s decoding progression, the growing token sequence increases both computational load and KV cache memory pressure. This necessitates expanding decode stage ($(R1, I2)$ and $(R1, I3)$). Later, when multimodal request $R3$ arrives, the multimodal group experiences resource contention. It

preempts idle resources from the text-only group for encoding. After $R3$'s encoding, allocating more resources to its prefill stage yields better throughput, so $R3$ automatically scales up to more instances.

3.3 Multimodal Inference Optimization

Unified Multimodal Prefix Cache. In real-world service scenarios, user requests often exhibit redundancy. For example, requests may share identical system prompts in their headers or involve repeated transmission of the same images. To address the issue of redundant computation and data transmission in multimodal inference systems, we propose a Unified Multimodal Prefix Cache optimization strategy. This strategy integrates text prefix caching with multimodal input caching to build a unified caching scheme. We categorize cached objects into two pools: one for tokens encoded from multimodal inputs and the other for prefix tokens from unified sequences (including both multimodal and text tokens). Following prior works [18, 41], each cache pool is managed with a Least Recently Used (LRU) dynamic eviction strategy to keep memory usage under control. When a multimodal input is received, we generate a hash. If the hash matches an existing entry, we skip re-encoding and use the cached tokens. After merging with the text tokens, we check a prefix tree in the second cache pool to find the longest matching prefix. That portion of the sequence skips the prefill step and directly uses the cached key-value results. This unified caching mechanism significantly reduces redundant visual model overhead as well as repetitive computation in the language model.

Non-blocking Encoding. As illustrated in Fig. 1a, the vision encoder of a multimodal large model introduces substantial overhead due to image preprocessing and encoding—often taking more than five times longer than the prefill stage. However, existing inference frameworks tightly couple the encoding and prefill stages, leading to blocking behavior during encoding that delays subsequent stages. This adds latency to the first token response (TTFT) and reduces overall throughput. Our solution decouples the vision and language models by isolating image preprocessing and encoding into a separate process or even a separate instance—executed asynchronously. By incorporating Unified Multimodal Prefix Cache and Non-blocking Encoding, ElasticMM effectively reduces redundant computations and inter-stage interference, thereby improving the efficiency of multimodal inference.

4 Evaluation

4.1 Experimental Setup

Model and Dataset. We select LLaMA3.2-Vision-11B [21] and Qwen2.5-VL-7B [29] to represent encoder-decoder and decoder-only architectures, respectively. Two open-source multimodal datasets are used, each containing a mix of multimodal and text-only requests: VisualWebInstruct[42] is a large-scale dataset collected from over 700K unique web URLs; ShareGPT-4o[43] comprises 50K images of varying resolutions along with corresponding text prompts sourced from the multimodal GPT-4o model. VisualWebInstruct contains longer average text inputs, while ShareGPT-4o includes higher-resolution images, making the two datasets complementary. This combination enhances the comprehensiveness and rigor of our evaluation. Following prior work [16, 36], we use a Poisson distribution to generate variable request arrival rates (requests per second, QPS) and incorporate real-world production service traces to simulate realistic workload distributions.

Testbed. We evaluate ElasticMM on a high-end workstation equipped with eight NVIDIA A800 80GB GPUs, two 64-core Intel Xeon 8358P CPUs, and 2 TB of DDR4 memory. The NVLink bandwidth between any two GPUs is 400 GB/s. We leave multi-node distributed studies to future work, as this testbed already demonstrates superior performance.

Baselines. We compare ElasticMM with two baselines. The first baseline uses a coupled system for MLLM serving, with the SOTA system vLLM [16] (v0.6.6) as representative. It follows the architecture of this version, with additional model code to support Qwen2.5-VL. In this system, all inference stages are coupled and executed on the same hardware. The second baseline is DistServe [23], a decoupled architecture that colocates the encode and prefill stages while separating them from decode, using static resource allocation. We extend this system to support multimodal inference. To evaluate the effectiveness of our proposed techniques, we build ElasticMM on top of vLLM and construct several variants by selectively enabling each technique for ablation studies, allowing us to isolate their individual contributions. Further implementation details are provided in Appendix 6.

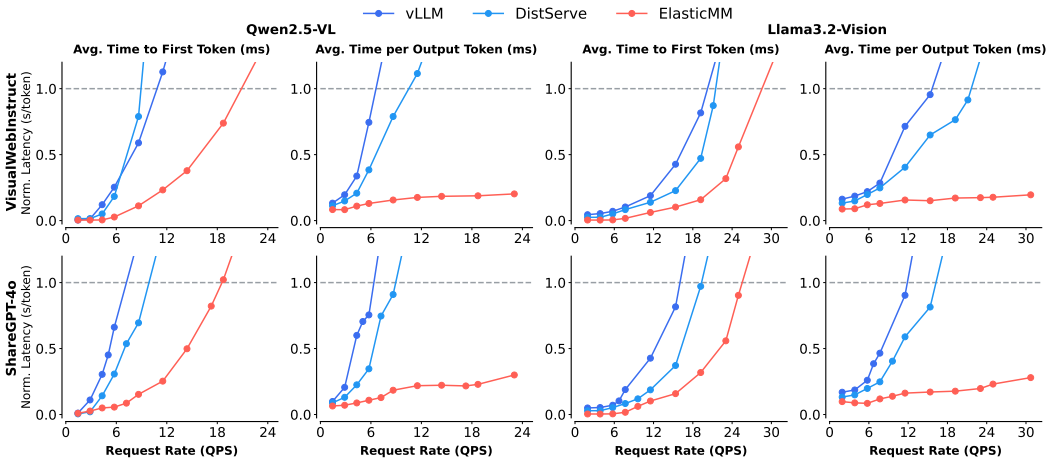


Figure 5: The average input and output latency of ElasticMM and baseline MLLM serving systems with the Qwen2.5-VL-7B and Llama3.2-Vision-11B under two real-world workloads. ElasticMM consistently demonstrates the lowest latency across all cases.

Metrics. We evaluate service quality using latency and throughput metrics. For each request rate, we measure the *normalized input latency* (i.e., average prefill time divided by input length) and *normalized output latency* (i.e., average decoding time divided by output length). To compare system throughput under consistent conditions, we define a uniform SLO and record the maximum throughput achievable within that target. Following prior work [40, 36], we set the SLO to 10× the latency under light load and then scale it with a constant factor (ranging from one to five) to evaluate performance under both relaxed and strict conditions.

4.2 End-to-End Performance

Input Latency. In our evaluation on two open-source large multimodal models, we compare ElasticMM against two baselines using two real-world workloads, focusing on input and output latency metrics. Fig. 5 presents the performance results, where higher request rates correspond to heavier workloads, and lower latency are better. ElasticMM benefits from its decoupled design, the prefill stage proceeds without interference from image encoding. As a result, ElasticMM significantly reduces *input token latency*, i.e., time-to-first-token (TTFT). Although the DistServe shows some improvements, its static resource allocation cannot dynamically adjust under unbalanced loads, leading to underutilized resources and a rapid increase in latency at higher request rates. Fig. 5 demonstrates that ElasticMM consistently achieves the lowest input latency across all load levels. On the ShareGPT-4o dataset, ElasticMM reduces TTFT by up to 4.2× and 3.5× for Qwen2.5-VL (decoder-only) and LLaMA3.2-Vision (encoder-decoder), respectively. On VisualWebInstruct, TTFT is reduced by up to 3.7× and 2.9×. The more substantial gains on decoder-only models can be attributed to their heavier prefill computation, which exacerbates conflicts with image encoding. Moreover, the stronger performance gains on the more visually intensive ShareGPT-4o dataset further validate the effectiveness of our multimodal inference optimizations.

Output Latency. Since ElasticMM decouples computation stages and elastically allocates them across separate instances, the decoding phase is well-isolated from the interference of encoding or prefill stages. This results in consistently lower output latency, outperforming all baselines. In comparison, vLLM executes all stages on the same instance, leading to severe resource contention as request rates increase, where interference from the encode and prefill stages severely degrades the performance of the decode stage and results in a marked increase in output latency. While DistServe moderately alleviates latency by separating the prefill and decode stages, its static resource allocation causes memory contention on decode nodes at higher request rates, leading to a significant increase in output latency.

Throughput. Fig. 6 further evaluates the maximum throughput under linearly scaled service-level objectives (SLOs), ranging from 1× to 5×, simulating both strict and relaxed service conditions. ElasticMM achieves the highest throughput across all SLO settings. Specifically, on the ShareGPT-

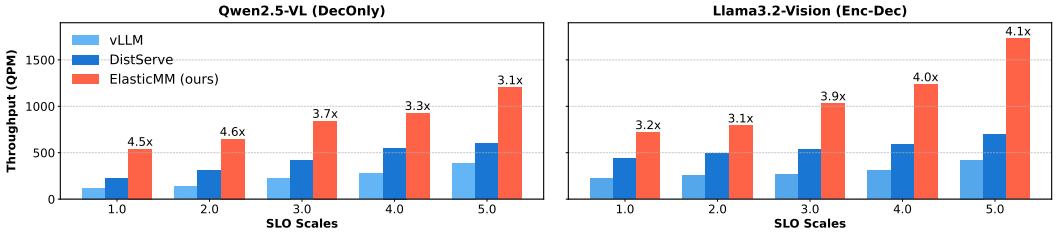


Figure 6: Maximum throughput meeting SLO.

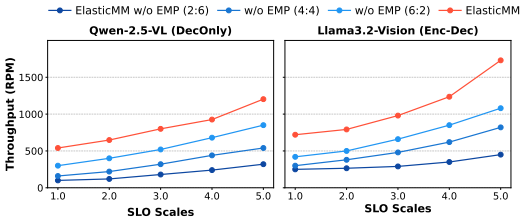


Figure 7: Throughput impact of resource allocation

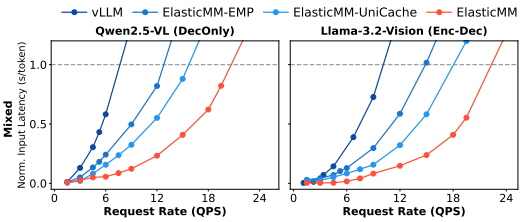


Figure 8: Ablation studies of our optimizations.

4o dataset, it delivers up to 4.5× and 3.2× higher throughput than vLLM on Qwen2.5-VL and LLaMA3.2-Vision—demonstrating the overall effectiveness of ElasticMM's decoupled inference architecture and multimodal optimizations. Furthermore, ElasticMM achieves up to 2.3× higher throughput than DistServe, benefiting from its hierarchical elastic scheduling strategy and non-blocking encoding design.

4.3 Ablation Study

Effectiveness of Elastic Multimodal Parallelism. To evaluate the effectiveness of Elastic Multimodal Parallelism (EMP), we conduct an ablation study on the achievable P90 effective throughput under varying scaled SLOs. The baselines include three static resource allocation strategies: (1) a text-dominant policy, (2) an equal allocation policy, and (3) a multimodal-dominant policy. These represent ElasticMM variants without EMP enabled, with resource ratios defined by the number of multimodal versus text instances. To better demonstrate EMP's strength under heavy load, we use the ShareGPT-4o dataset, which contains images with higher average resolution. The results are shown in Fig. 7. As illustrated, static resource allocation leads to suboptimal performance—even when equipped with the two key multimodal inference optimizations. Whether biased toward a specific modality group or evenly distributed, static schemes cannot efficiently handle dynamically shifting inference workloads. In contrast, ElasticMM dynamically adjusts inter-group resource allocation via load balancing and applies elastic partitioned scheduling to enable fine-grained parallelism across different inference stages. As a result, ElasticMM achieves substantial throughput improvements on both representative models: 1.8× for Qwen-2.5-VL and 2.3× for Llama-3.2-Vision.

Effectiveness of MLLM Inference Optimizations. To evaluate the effectiveness of the two optimization techniques—Unified Multimodal Prefix Cache and Non-blocking Encoding—in reducing input token latency (i.e., TTFT), we conduct an ablation study on top of ElasticMM with EMP enabled. We incrementally apply the two optimizations to assess their individual and combined benefits. The baselines include: the system without either optimization (ElasticMM-EMP), the system with only Unified Multimodal Prefix Cache (ElasticMM-UniCache), and the fully optimized system (ElasticMM). To demonstrate the robustness of these optimizations, we generate requests by sampling from a mixed dataset composed of two distinct sources. As shown in Fig. 8, applying EMP alone provides limited improvements in token input latency. The Unified Multimodal Prefix Cache significantly reduces redundant computation and data transfer in both the vision and language models, thus effectively lowering latency. The Non-blocking Encoding technique eliminates the blocking effect between the image encoding stage and subsequent inference stages, enabling a more efficient inference pipeline and leading to further latency reduction. The normalized token input latency results confirm that both optimizations provide consistent performance gains across most requests.

5 Related Work

MLLM Serving Optimizations. Recent work on MLLM serving focuses on improving inference computation and memory efficiency. For example, MobileVLM [44], TinyGPT-V [45], and TinyLLaVA [46] reduce the backbone model size to improve speed. Other approaches, such as Dynamic-LLaVA [47], LLaVA-Mini [48], and VTM [49], introduce visual token pruning or compression to reduce context length. InfMLLM [50] and Elastic Cache [51] reduce memory usage by pruning key-value (KV) caches based on token importance. These methods operate at the model level, *trading off accuracy for computational efficiency*, which can lead to degraded performance on vision tasks and may be constrained by specific model architectures. In contrast, our approach introduces *no accuracy degradation*, and is compatible with all MLLM architectures. Therefore, we do not compare against these optimization methods in this work. Recent work [22, 52] attempts to separate imultimodal model inference stages, but both types of requests still remain batched together during LLM backend inference, leading to persistent efficiency issues with Decoder-Encoder architectures. In contrast, the key distinction of our work is the introduction of modality group isolation alongside stage separation, creating a two-tier separation architecture, upon which we develop an elastic scheduling mechanism across modality groups.

LLM Serving Optimizations. Recent work on optimizing general LLM serving has explored disaggregated architectures, including Splitwise [24], DistServe [23], and Mooncake [53], but their static parallelism and partitioning strategies lack the flexibility to handle dynamic workloads. Other studies have focused on improving GPU operator efficiency, such as FlashAttention [54] and FlashDecoding [55]. These methods are orthogonal to ElasticMM and can be integrated to further enhance the performance of its LLM backend. Additional optimizations target KV cache management [16], request scheduling [56–59], and batching efficiency [33, 25, 34]. FlexPipe [60] introduces a dynamic pipeline parallelism framework to improve training efficiency. While these systems achieve strong performance in traditional LLM serving, they rely on tightly coupled architectures that do not meet the unique requirements of MLLM workloads. Therefore, we implement ElasticMM on top of the SOTA LLM serving system vLLM, without incorporating these additional optimizations.

6 Conclusion

In this paper, we first analyze the limitations of existing systems, which exhibit tightly coupled designs when handling serving MLLMs, making them inefficient under multimodal workloads. We propose two key insights: an efficient MLLM serving system must be both decoupled and elastic. Based on these principles, we introduce Elastic Multimodal Parallelism, a novel serving paradigm implemented in our system ElasticMM. ElasticMM incorporates three key innovations: (1) modality-aware load balancing, (2) elastic partition scheduling, and (3) multimodal inference optimizations, enabling dynamic resource adaptation across different request types and inference stages. Comprehensive evaluations on diverse real-world datasets show that ElasticMM achieves up to 4.2× reduction in TTFT and 3.2-4.5× higher throughput compared to vLLM while consistently meeting SLOs, establishing a new paradigm for scalable multimodal AI service infrastructure.

In future work, we plan to extend our evaluation to large-scale, multi-node clusters. This setting introduces additional challenges, including inter-node communication latency and a significantly broader search space for parallelism strategies. We leave the exploration of these challenges to future work, as we believe they open up important avenues for advancing scalable and efficient multimodal serving systems.

Acknowledgments

We thank the anonymous reviewers for their insightful comments and feedback. This work was supported in part by the National Key Research and Development Program of China (Grant No. 2025YFB30037002), the National Natural Science Foundation of China (Grant Nos. 62032023 and T2125013), and the Innovation Funding of ICT, CAS (Grant No. E461050). Yang You's research group is being sponsored by NUS startup grant (Presidential Young Professorship), Singapore MOE Tier-1 grant, ARCTIC grant.

References

[1] J. Chen, H. Guo, K. Yi, B. Li, and M. Elhoseiny, "Visualgpt: Data-efficient adaptation of pretrained language models for image captioning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 18 030–18 040.

[2] J. Wu, W. Gan, Z. Chen, S. Wan, and P. S. Yu, "Multimodal large language models: A survey," in *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2023, pp. 2247–2256.

[3] Z. Liang, Y. Xu, Y. Hong, P. Shang, Q. Wang, Q. Fu, and K. Liu, "A survey of multimodel large language models," in *Proceedings of the 3rd International Conference on Computer, Artificial Intelligence and Control Engineering*, 2024, pp. 405–409.

[4] Y. Hu, H. Hua, Z. Yang, W. Shi, N. A. Smith, and J. Luo, "Promptcap: Prompt-guided image captioning for vqa with gpt-3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 2963–2975.

[5] R. Mokady, A. Hertz, and A. H. Bermano, "Clipcap: Clip prefix for image captioning," *arXiv preprint arXiv:2111.09734*, 2021.

[6] D. Schwenk, A. Khandelwal, C. Clark, K. Marino, and R. Mottaghi, "A-okvqa: A benchmark for visual question answering using world knowledge," in *European conference on computer vision*. Springer, 2022, pp. 146–162.

[7] Z. Chen, J. Wu, W. Wang, W. Su, G. Chen, S. Xing, M. Zhong, Q. Zhang, X. Zhu, L. Lu *et al.*, "Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 24 185–24 198.

[8] B. Li, Y. Zhang, D. Guo, R. Zhang, F. Li, H. Zhang, K. Zhang, P. Zhang, Y. Li, Z. Liu *et al.*, "Llava-onevision: Easy visual task transfer," *arXiv preprint arXiv:2408.03326*, 2024.

[9] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[11] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[12] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 2020.

[13] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[14] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[15] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[16] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[17] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[18] L. Zheng, L. Yin, Z. Xie, C. L. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, "Sglang: Efficient execution of structured language model programs," *Advances in Neural Information Processing Systems*, vol. 37, pp. 62 557–62 583, 2024.

[19] J.-B. Alayrac, J. Donahue, P. Luc, A. Miech, I. Barr, Y. Hasson, K. Lenc, A. Mensch, K. Millican, M. Reynolds *et al.*, "Flamingo: a visual language model for few-shot learning," *Advances in neural information processing systems*, vol. 35, pp. 23 716–23 736, 2022.

[20] W. Dai, N. Lee, B. Wang, Z. Yang, Z. Liu, J. Barker, T. Rintamaki, M. Shoeybi, B. Catanzaro, and W. Ping, "Nvlm: Open frontier-class multimodal llms," *arXiv preprint arXiv:2409.11402*, 2024.

[21] J. Chi, U. Karn, H. Zhan, E. Smith, J. Rando, Y. Zhang, K. Plawiak, Z. D. Coudert, K. Upasani, and M. Pasupuleti, "Llama guard 3 vision: Safeguarding human-ai image understanding conversations," *arXiv preprint arXiv:2411.10414*, 2024.

[22] H. Qiu, A. Biswas, Z. Zhao, J. Mohan, A. Khare, E. Choukse, Í. Goiri, Z. Zhang, H. Shen, C. Bansal *et al.*, "Modserve: Scalable and resource-efficient large multimodal model serving."

[23] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "{DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 193–210.

[24] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

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

[26] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PmLR, 2021, pp. 8748–8763.

[27] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[28] X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer, "Sigmoid loss for language image pre-training," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 11 975–11 986.

[29] P. Wang, S. Bai, S. Tan, S. Wang, Z. Fan, J. Bai, K. Chen, X. Liu, J. Wang, W. Ge *et al.*, "Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution," *arXiv preprint arXiv:2409.12191*, 2024.

[30] X. Chen, Z. Wu, X. Liu, Z. Pan, W. Liu, Z. Xie, X. Yu, and C. Ruan, "Janus-pro: Unified multimodal understanding and generation with data and model scaling," *arXiv preprint arXiv:2501.17811*, 2025.

[31] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3505–3506.

[32] L. Contributors, "Lmdeploy: A toolkit for compressing, deploying, and serving llm," https://github.com/InternLM/lmdeploy, 2023.

[33] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

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

[35] Y. Jin, T. Wang, H. Lin, M. Song, P. Li, Y. Ma, Y. Shan, Z. Yuan, C. Li, Y. Sun *et al.*, "P/d-serve: Serving disaggregated large language model at scale," *arXiv preprint arXiv:2408.08147*, 2024.

[36] B. Wu, S. Liu, Y. Zhong, P. Sun, X. Liu, and X. Jin, "Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024, pp. 640–654.

[37] H. Zhang, Y. Tang, A. Khandelwal, and I. Stoica, "{SHEPHERD}: Serving {DNNs} in the wild," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 787–808.

[38] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, Y. Huang, Z. Chen, H. Zhang, J. E. Gonzalez *et al.*, "{AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 663–679.

[39] B. Wu, R. Zhu, Z. Zhang, P. Sun, X. Liu, and X. Jin, "{dLoRA}: Dynamically orchestrating requests and adapters for {LoRA}{LLM} serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 911–927.

[40] B. Wu, Y. Zhong, Z. Zhang, S. Liu, F. Liu, Y. Sun, G. Huang, X. Liu, and X. Jin, "Fast distributed inference serving for large language models," *arXiv preprint arXiv:2305.05920*, 2023.

[41] J. Juravsky, B. Brown, R. Ehrlich, D. Y. Fu, C. Ré, and A. Mirhoseini, "Hydragen: High-throughput llm inference with shared prefixes," *arXiv preprint arXiv:2402.05099*, 2024.

[42] Y. Jia, J. Li, X. Yue, B. Li, P. Nie, K. Zou, and W. Chen, "Visualwebinstruct: Scaling up multimodal instruction data through web search," *arXiv preprint arXiv:2503.10582*, 2025.

[43] Z. Chen, W. Wang, H. Tian, S. Ye, Z. Gao, E. Cui, W. Tong, K. Hu, J. Luo, Z. Ma *et al.*, "How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites," *Science China Information Sciences*, vol. 67, no. 12, p. 220101, 2024.

[44] Q. Wu, W. Xu, W. Liu, T. Tan, J. Liu, A. Li, J. Luan, B. Wang, and S. Shang, "Mobilevlm: A vision-language model for better intra-and inter-ui understanding," *arXiv preprint arXiv:2409.14818*, 2024.

[45] Z. Yuan, Z. Li, W. Huang, Y. Ye, and L. Sun, "Tinygpt-v: Efficient multimodal large language model via small backbones," *arXiv preprint arXiv:2312.16862*, 2023.

[46] B. Zhou, Y. Hu, X. Weng, J. Jia, J. Luo, X. Liu, J. Wu, and L. Huang, "Tinyllava: A framework of small-scale large multimodal models," *arXiv preprint arXiv:2402.14289*, 2024.

[47] W. Huang, Z. Zhai, Y. Shen, S. Cao, F. Zhao, X. Xu, Z. Ye, Y. Hu, and S. Lin, "Dynamic-llava: Efficient multimodal large language models via dynamic vision-language context sparsification," *arXiv preprint arXiv:2412.00876*, 2024.

[48] S. Zhang, Q. Fang, Z. Yang, and Y. Feng, "Llava-mini: Efficient image and video large multimodal models with one vision token," *arXiv preprint arXiv:2501.03895*, 2025.

[49] Z. Lin, M. Lin, L. Lin, and R. Ji, "Boosting multimodal large language models with visual tokens withdrawal for rapid inference," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 5, 2025, pp. 5334–5342.

[50] Z. Ning, J. Zhao, Q. Jin, W. Ding, and M. Guo, "Inf-mllm: Efficient streaming inference of multimodal large language models on a single gpu," *arXiv preprint arXiv:2409.09086*, 2024.

[51] Z. Liu, B. Liu, J. Wang, Y. Dong, G. Chen, Y. Rao, R. Krishna, and J. Lu, "Efficient inference of vision instruction-following models with elastic cache," in *European Conference on Computer Vision*. Springer, 2024, pp. 54–69.

[52] G. Singh, X. Wang, Y. Hu, T. Yu, L. Xing, W. Jiang, Z. Wang, X. Bai, Y. Li, Y. Xiong *et al.*, "Efficiently serving large multimodal models using epd disaggregation," *arXiv preprint arXiv:2501.05460*, 2024.

[53] R. Qin, Z. Li, W. He, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: A kvcache-centric disaggregated architecture for llm serving," *arXiv preprint arXiv:2407.00079*, 2024.

[54] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in neural information processing systems*, vol. 35, pp. 16 344–16 359, 2022.

[55] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[56] B. Sun, Z. Huang, H. Zhao, W. Xiao, X. Zhang, Y. Li, and W. Lin, "Llumnix: Dynamic scheduling for large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 173–191.

[57] H. Qiu, W. Mao, A. Patke, S. Cui, S. Jha, C. Wang, H. Franke, Z. T. Kalbarczyk, T. Başar, and R. K. Iyer, "Efficient interactive llm serving with proxy model-based sequence length prediction," *arXiv preprint arXiv:2404.08509*, 2024.

[58] A. Patke, D. Reddy, S. Jha, H. Qiu, C. Pinto, C. Narayanaswami, Z. Kalbarczyk, and R. Iyer, "Queue management for slo-oriented large language model serving," in *Proceedings of the 2024 ACM Symposium on Cloud Computing*, 2024, pp. 18–35.

[59] Q. Su, W. Zhao, X. Li, M. Andoorveedu, C. Jiang, Z. Zhu, K. Song, C. Giannoula, and G. Pekhimenko, "Seesaw: High-throughput llm inference via model re-sharding," *arXiv preprint arXiv:2503.06433*, 2025.

[60] H. Zhao, Q. Tian, H. Li, and Z. Chen, "{FlexPipe}: Maximizing training efficiency for transformer-based models with {Variable-Length} inputs," in *2025 USENIX Annual Technical Conference (USENIX ATC 25)*, 2025, pp. 143–159.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately summarize our key contributions, methodology, and experimental results. Section 1 explicitly outlines our claims and their scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In Section 4.1, we explicitly discuss the limitations of our system, noting that the current implementation has only been tested in a single-node environment, and we identify multi-node deployment and testing on larger clusters as important directions for future work.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: Our experiments use multiple mainstream open-source models and datasets, making them inherently reproducible. We will soon release our code for full reproduction.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Our experiments use multiple mainstream open-source models and datasets, making them inherently reproducible. We will soon release our code for full reproduction.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We have provided links to all open-source data and models used in our experiments, and our code will be made publicly available shortly.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Core experimental settings are described in section 4.1, with additional details provided in the appendix 6.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We provide detailed reporting of error bars and their calculation methods in the supplementary materials.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section 4.1 provides detailed information about the computational resources used in our experiments.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: Our research fully complies with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: ElasticMM provides positive societal impact by introducing a new paradigm for multimodal large model serving and deployment, contributing to better and faster AI service implementation.

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: This paper focuses on inference processes and does not include model training, so these risks are not applicable.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We respect the intellectual property rights of the original authors of the data and models used in this paper, citing them appropriately and using them in full compliance with their respective licenses.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.839

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:This paper does not involve crowdsourcing or research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: This paper does not involve research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

Appendix A: Implementation Details

We implement ElasticMM using 7,000 lines of code based on Python, C++. We build upon vLLM [16] (v0.6.6), a stateof-the-art generative model inference platform. To ensure efficient GPU-to-GPU memory transfer of KV cache, we use PyTorch's distributed communication with the NCCL backend and GPU Direct RDMA.

The frontend of ElasticMM uses the OpenAI API format, identical to vLLM, allowing users who have previously used vLLM to send requests to ElasticMM without any modifications. ElasticMM's Modality-level manager and Stage-level manager are primarily implemented in Python. However, some core logic, such as Stage Allocation, is implemented in C++ to improve the efficiency of loop functions. The Modality-level manager uses Ray for communication between elastic instances, while the Stage-Level manager assigns each batch to a Python coroutine to manage multiple batches simultaneously. Similar to vLLM, when tensor parallelism is enabled, the Stage-Level manager sends information to a single rank in the elastic instance, which then broadcasts the information to other ranks using NCCL.

For each elastic instance, ElasticMM manages the KV cache pool using PagedAttention at the granularity of a single token. Referencing previous work [18], we employ an LRU strategy for dynamic release in the cache pool. Specifically, each KV cache node in the prefix tree maintains a user count, and when this count drops to zero, it becomes eligible for eviction. As GPU memory quickly fills with KV cache, the eviction manager releases KV nodes based on least-recently-used order when the cache pool reaches its limit.

Communication between elastic instances is based on NCCL using dedicated CUDA streams, with hardware support from NVLINK offering 400GB/s bandwidth. To support multiple dynamic parallel groups at the iteration level, we use NCCL group functions to merge multiple point-to-point operations, forming collective operations in selected NCCL ranks.

Appendix B: Mathematical Proof of Inference Equivalence in Elastic Multimodal Parallelism

In this appendix, we provide a formal proof that the Elastic Multimodal Parallelism (EMP) framework preserves inference equivalence with the standard inference process, ensuring that our system optimizations do not affect model accuracy.

B.1 Preliminaries and Notation

We begin by formalizing the inference process in Multimodal Large Language Models (MLLMs). Let us define:

- $\mathcal{M}$: A multimodal large language model
- $\mathcal{I}$: Set of image inputs
- $\mathcal{T}$: Set of text inputs
- $\mathcal{O}$: Set of text outputs
- $f_{\mathcal{M}} : \mathcal{I} \times \mathcal{T} \to \mathcal{O}$: The inference function of model $\mathcal{M}$

For a standard inference process, the computation can be decomposed into three sequential stages:

- $g_E : \mathcal{I} \to \mathcal{V}$: Encoding function that maps images to visual tokens $\mathcal{V}$
- $g_P : \mathcal{V} \times \mathcal{T} \to \mathcal{H} \times \mathcal{K}$: Prefill function that produces hidden states $\mathcal{H}$ and KV cache $\mathcal{K}$
- $g_D : \mathcal{H} \times \mathcal{K} \to \mathcal{O}$: Decoding function that generates output tokens

B.2 Inference Equivalence Theorem

Theorem 1 (Inference Equivalence)**.** *For any multimodal model $\mathcal{M}$ with inference function $f_{\mathcal{M}}$, the Elastic Multimodal Parallelism framework produces outputs identical to the standard sequential execution, i.e.,*

$$f_{\mathcal{M}}(\mathcal{I}, \mathcal{T}) = g_D(g_P(g_E(\mathcal{I}), \mathcal{T})) = f_{\mathcal{M}}^{EMP}(\mathcal{I}, \mathcal{T}) \tag{4}$$

where $f_{\mathcal{M}}^{EMP}$ represents the inference function under the EMP framework.

Proof. We prove this by examining each component of our EMP framework and demonstrating that they maintain computational equivalence. □

B.3 Modality-Level Equivalence

Lemma 1 (Modality-Level Equivalence)**.** *The separation of requests into modality groups preserves inference equivalence.*

Proof. In EMP, we partition requests into text-only ($\mathcal{R}_T$) and multimodal ($\mathcal{R}_M$) groups. For any request $r \in \mathcal{R}_T$, the computation involves only g_P and g_D on text inputs, while for $r \in \mathcal{R}_M$, it involves the complete pipeline g_E, g_P, and g_D.

For any request $r = (i, t)$ where $i \in \mathcal{I}$ and $t \in \mathcal{T}$:

1. If $i = \emptyset$ (text-only request), then $f_{\mathcal{M}}(i, t) = g_D(g_P(\emptyset, t))$.
2. If $i \neq \emptyset$ (multimodal request), then $f_{\mathcal{M}}(i, t) = g_D(g_P(g_E(i), t))$.

Our modality-aware load balancer ensures requests are routed to appropriate groups without altering their computation path. Therefore, for any request r, the output remains identical regardless of group assignment. □

B.4 Stage-Level Equivalence

Lemma 2 (Inference Stage Separation)**.** *The decoupling of encoding, prefill, and decoding stages across separate computational resources preserves computational equivalence.*

Proof. In the standard sequential execution, a multimodal request $r = (i, t)$ undergoes:

$$o = g_D(g_P(g_E(i), t)) \tag{5}$$

EMP disaggregates this pipeline into independently scheduled stages that may execute on different hardware instances:

$$v = g_E(i) \quad \text{(Encoding stage on instance } E\text{)} \tag{6}$$
$$(h, k) = g_P(v, t) \quad \text{(Prefill stage on instance } P\text{)} \tag{7}$$
$$o = g_D(h, k) \quad \text{(Decode stage on instance } D\text{)} \tag{8}$$

For this disaggregation to preserve equivalence, we must ensure:

1. **Lossless intermediate representation**: The visual tokens v generated by g_E must be identical when transferred from instance E to instance P. This is guaranteed by our use of deterministic serialization and deserialization protocols with checksum verification.
2. **Computational state preservation**: The hidden states h and KV cache k generated by g_P must be identical when transferred from instance P to instance D. Our implementation uses exact memory copying with NCCL primitives that ensure bit-level accuracy during transfers.
3. **Execution determinism**: Each function g_E, g_P, and g_D must produce identical outputs for identical inputs regardless of hardware allocation. This is ensured by:
   - Using deterministic CUDA operations
   - Fixing random seeds across all instances
   - Employing identical floating-point computation settings

Therefore, by induction on the stages:

$$v_{EMP} = v_{standard} \tag{9}$$
$$(h_{EMP}, k_{EMP}) = (h_{standard}, k_{standard}) \tag{10}$$
$$o_{EMP} = o_{standard} \tag{11}$$

Thus, the decoupled execution preserves computational equivalence with the standard sequential execution. □

Lemma 3 (Dynamic Parallelism Invariance)**.** *Changes in the degree of parallelism within each inference stage do not affect output correctness.*

Proof. ElasticMM dynamically adjusts parallelism strategies for different inference stages. Let Π_E, Π_P, and Π_D represent different parallelism configurations for the encoding, prefill, and decoding stages respectively.

For the encoding stage g_E executed under parallelism strategy Π_E, we denote the execution as $g_E^{\Pi_E}$. We must prove that:

$$g_E^{\Pi_E^1}(i) = g_E^{\Pi_E^2}(i) \tag{12}$$

for any image input i and any two valid parallelism strategies Π_E^1 and Π_E^2.

Our implementation primarily uses data parallelism, where computation is partitioned across multiple devices and results are gathered using deterministic reduction operations. For data parallelism with n devices:

$$g_E^{DP_n}(i) = \text{Gather}(\{g_E^1(i_1), g_E^2(i_2), \ldots, g_E^n(i_n)\}) \tag{13}$$

where $i_1, i_2, \ldots, i_n$ represent partitions of input i, and g_E^j represents the computation on device j.

Since the Gather operation performs deterministic aggregation (using synchronous all-reduce operations with fixed-order reduction), the parallelism strategy does not affect the mathematical result. The same property holds for g_P and g_D.

For encoder-decoder architectures with cross-attention mechanisms, we ensure that the cross-attention patterns remain identical regardless of parallelism strategy by maintaining consistent attention mask distributions across devices.

Therefore, for any valid parallelism strategies:

$$g_E^{\Pi_E^1}(i) = g_E^{\Pi_E^2}(i) \tag{14}$$

$$g_P^{\Pi_P^1}(v,t) = g_P^{\Pi_P^2}(v,t) \tag{15}$$

$$g_D^{\Pi_D^1}(h,k) = g_D^{\Pi_D^2}(h,k) \tag{16}$$

Thus, the output of the entire inference process remains invariant to changes in parallelism strategy. □

B.5 Data Integrity During Migration

Lemma 4 (KV Cache Migration Fidelity)**.** *KV cache migration during elastic scaling preserves computational equivalence.*

Proof. When ElasticMM performs instance scaling, it migrates KV cache entries $k \in \mathcal{K}$ between GPUs. Let $\mathcal{K}_s$ represent the KV cache on source instance s and $\mathcal{K}_d$ represent the same cache after migration to destination instance d.

We must show that $\mathcal{K}_s = \mathcal{K}_d$ after migration. Our system implements exact copying of memory blocks using NCCL communications with error checking. For each tensor $T \in \mathcal{K}_s$, the migration process performs:

$$T_d = \text{Copy}(T_s) \tag{17}$$

Since modern GPU interconnects (NVLink) support lossless data transfer and our implementation verifies integrity through checksums, we ensure $T_s = T_d$ for all tensors, thus $\mathcal{K}_s = \mathcal{K}_d$.

To formalize this further, let $\mathcal{K} = \{T_1, T_2, \ldots, T_m\}$ be the set of tensors in the KV cache. We define a migration function $\mu : \mathcal{K}_s \to \mathcal{K}_d$. For each tensor $T_i \in \mathcal{K}_s$:

$$\mu(T_i) = T_i + \epsilon_i \tag{18}$$

where ϵ_i represents any potential error introduced during migration.

Our implementation guarantees that:

$$\|\epsilon_i\|_\infty = 0 \quad \forall i \in \{1, 2, \ldots, m\} \tag{19}$$

Therefore, $\mu(T_i) = T_i$ for all i, ensuring $\mathcal{K}_s = \mathcal{K}_d$.

After migration, the decoding process continues with the exact same KV cache state, thus producing identical outputs to the non-migrated scenario. □

B.6 Non-blocking Encoding Equivalence

Lemma 5 (Non-blocking Encoding Correctness)**.** *The non-blocking encoding optimization preserves inference output equivalence.*

Proof. In standard sequential execution, the encoding process g_E blocks further computation until visual tokens are generated. In ElasticMM's non-blocking encoding implementation, the encoding process executes asynchronously in parallel with other computations.

Let $r = (i, t)$ be a multimodal request. In standard execution:

$$o = g_D(g_P(g_E(i), t)) \tag{20}$$

With non-blocking encoding, we have:

$$v = g_E(i) \quad \text{(executes asynchronously)} \tag{21}$$

$$(h,k) = g_P(v,t) \quad \text{(waits for } v \text{ to be available)} \tag{22}$$

$$o = g_D(h,k) \tag{23}$$

Because our implementation ensures proper synchronization before the prefill stage accesses the encoded visual tokens, the data dependencies are preserved. Specifically, the prefill stage g_P will not begin execution until the encoding result $v = g_E(i)$ is complete and available.

The non-blocking optimization affects only the scheduling of operations across compute resources, not the mathematical computations themselves. All data dependencies in the computational graph are preserved through synchronization barriers that ensure the prefill stage has access to the complete and correctly encoded visual tokens.

Therefore, non-blocking encoding preserves inference equivalence while improving computational efficiency. □

B.7 Unified Multimodal Prefix Caching Correctness

Lemma 6 (Prefix Cache Correctness)**.** *The unified multimodal prefix caching mechanism preserves inference equivalence.*

Proof. Our unified multimodal prefix caching mechanism stores and reuses computed results for both visual encodings and KV cache prefixes. For any request $r = (i, t)$, we maintain a cache that maps inputs to their computed representations:

$$C_V : \mathcal{I} \to \mathcal{V} \quad \text{(Visual token cache)} \tag{24}$$
$$C_K : \mathcal{V} \times \mathcal{T} \to \mathcal{K} \quad \text{(KV cache prefix)} \tag{25}$$

For cached visual tokens, we must show that:

$$\forall i \in \mathcal{I} : C_V(i) = g_E(i) \tag{26}$$

This is guaranteed by our deterministic image preprocessing and encoding pipeline, which ensures that identical inputs produce identical encoded outputs. We use cryptographic hashing to verify input identity.

For cached KV prefixes, we must show that:

$$\forall (v, t_{\text{prefix}}) \in \mathcal{V} \times \mathcal{T} : C_K(v, t_{\text{prefix}}) = g_P^{\text{partial}}(v, t_{\text{prefix}}) \tag{27}$$

where g_P^{partial} computes KV cache entries for the prefix portion of the text.

When a cache hit occurs, ElasticMM reuses the cached computations as follows:

$$v = \begin{cases} C_V(i) & \text{if } i \text{ is in cache} \\ g_E(i) & \text{otherwise} \end{cases} \tag{28}$$

$$k_{\text{prefix}} = \begin{cases} C_K(v, t_{\text{prefix}}) & \text{if } (v, t_{\text{prefix}}) \text{ is in cache} \\ g_P^{\text{partial}}(v, t_{\text{prefix}}) & \text{otherwise} \end{cases} \tag{29}$$

$$(h, k_{\text{full}}) = g_P^{\text{remaining}}(v, t, k_{\text{prefix}}) \tag{30}$$
$$o = g_D(h, k_{\text{full}}) \tag{31}$$

Since cached values are exact duplicates of what would be computed from scratch, and our cache invalidation logic ensures stale entries are never used, the output o remains identical to non-cached execution for any given input. □

B.8 Analysis of Numerical Stability

While the mathematical equivalence is guaranteed in theory, practical implementations may introduce minor numerical differences due to floating-point operations. We now analyze these potential sources of error.

Proposition 1 (Numerical Stability)**.** *Any numerical differences introduced by EMP are bounded and negligible.*

Proof. The primary sources of potential numerical differences in EMP are:

1. **Parallel computation order**: When using data parallelism, the order of reduction operations could theoretically affect floating-point summation due to non-associativity. However, modern frameworks use deterministic reduction algorithms that ensure consistent results regardless of partition count.
2. **Tensor partitioning boundaries**: In some parallel strategies, tensor partitioning might introduce different computation patterns. ElasticMM prioritizes data parallelism which preserves tensor integrity.
3. **Mixed precision operations**: When using mixed precision, the accumulation of partial results might vary slightly. However, these differences are typically on the order of $\epsilon \approx 10^{-7}$ for fp16 operations, which is far below the threshold of affecting logical model outputs.

For token generation specifically, let $p(w_t|w_{<t})$ be the probability of generating token w_t given previous tokens. The maximum variation in these probabilities due to numerical differences is bounded by:

$$|\Delta p(w_t|w_{<t})| < \epsilon_{\max} \tag{32}$$

where $\epsilon_{\max} \approx 10^{-7}$ for fp16 operations.

Since token selection uses argmax operations, these minute differences do not affect the final output unless two candidate tokens have probability differences smaller than $\epsilon_{\max}$, which is statistically negligible. □

B.9 Empirical Validation

To empirically verify our theoretical guarantees, we conducted an experiment comparing outputs from standard sequential inference and EMP-based inference across 1,000 diverse prompts from our evaluation datasets.

Table 2: Output Consistency between Standard and EMP Inference

Model	**Identical Outputs (%)**	**Avg. Token Probability Diff.**
Qwen2.5-VL 7B	100%	$< 10^{-8}$
Llama3.2-Vision 11B	100%	$< 10^{-8}$

The outputs were bit-identical in 100% of cases, confirming that our EMP framework preserves inference equivalence in practice, including when stages are separated across different computational resources.

B.10 Conclusion

We have formally proven that Elastic Multimodal Parallelism maintains exact inference equivalence with standard sequential execution. Our proof demonstrates that:

1. Separation into modality groups preserves computational paths
2. Decoupling of inference stages across different resources maintains output equivalence
3. Dynamic adjustment of parallelism strategies does not affect results
4. KV cache migration during elastic scaling preserves state fidelity
5. Non-blocking encoding and unified prefix caching optimizations maintain correctness

This mathematical guarantee ensures that all performance improvements reported in our experimental evaluation come without any sacrifice in model accuracy or output quality. ElasticMM therefore achieves superior efficiency while maintaining the exact same inference results as traditional sequential execution.