

Toward Parallel Serving for Vision-Language Models via Modal Decoupling and Scheduling

Yijia Yang*, Yubo Deng*, Yida Wang*, Yuanchao Xu*[†], Keni Qiu*,

*College of Information Engineering, Capital Normal University, Beijing, China

[†]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
{xuyuanchao, qiukn}@cnu.edu.cn

Abstract—Vision-Language Models (VLMs) have demonstrated strong performance in tasks such as image captioning and visual question answering. Under mixed workloads, however, the differing inference pipelines for text-only and multimodal requests create heterogeneity that existing serving systems fail to optimize—leading to high latency and poor fairness. We propose *DuetInfer*, a modality-aware serving framework that enhances single-GPU serving efficiency for VLMs through three key contributions: (i) parallel computation enabled by preprocessing parallelism and decoupled vision-language execution, (ii) a shared memory manager that eliminates weight redundancy and supports efficient encoder cache sharing, and (iii) a fairness-aware scheduler that reduces delays for multimodal requests without penalizing text-only ones. Implemented within vLLM and evaluated on realistic workloads, *DuetInfer* reduces P99 TTFT by up to 33.7% and end-to-end latency by up to 20%.

I. INTRODUCTION

Traditional large language models (LLMs) such as GPT-3 [1], BERT [2], and RoBERTa [3], are often inadequate for scenarios that demand the integration of heterogeneous data sources and complex reasoning. To overcome these limitations, researchers have developed Vision-Language Models (VLMs) [4] that combine multiple modalities, including text, images, videos, and beyond. With the debut of advanced multimodal systems such as GPT-4 [5] and Gemini [6], which exhibit impressive understanding and generation capabilities, VLMs have shown great potential in domains such as image captioning [7] and visual question answering [8]. However, many deployments of VLMs—ranging from cloud services to resource-constrained environments such as single-GPU servers or edge devices—require efficient serving under mixed workloads, making single-GPU optimization an important and practical concern.

Unlike conventional LLMs that process text-only inputs, VLMs typically serve two distinct types of requests: text-only and multimodal (e.g., text-image or text-video) [9], [10]. These request types follow markedly different computational paths, leading to significant disparities in their execution demands. Multimodal requests incur considerable preprocessing overhead as input images or videos are transformed into high-dimensional tensor representations, followed by compute-intensive vision encoding using models such as Vision Transformers (ViT) [11] to produce visual tokens. In contrast, text-only requests undergo minimal preprocessing, consisting primarily of tokenization. The two request types converge only

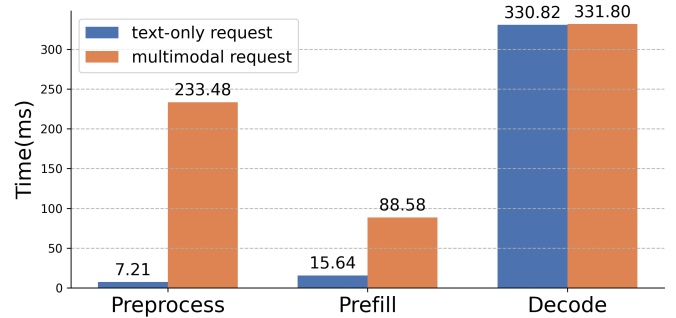


Fig. 1. Latency breakdown for different request types.

in the language modeling stage, where both text and vision tokens are processed jointly by the language model.

Existing serving engines such as vLLM [12], however, are not optimized for such heterogeneous workloads. They treat all requests uniformly, resulting in inefficiencies. To learn more details, we execute two requests of identical input length—multimodal request and text-only request—and obtained the same output token count. As shown in Fig. 1, the multimodal request shows 32.4× higher latency in the preprocessing stage, which arises from additional vision preprocessing, and a 5.8× longer prefill phase, mainly attributable to the cost of vision encoding. It can be seen that there is a significant difference in latency between multimodal requests and plain text-only requests. Based on experimental analysis and open-source code, we found that the scheduling of vLLM has at least the following shortcomings. Lightweight text-only requests are forced to wait until multimodal requests at the head of the queue finish processing, leading to severe head-of-line blocking. Moreover, the sequential execution of vision encoding and language modeling further exacerbates the problem, as the vision encoder computation directly stalls the progress of the language model.

To address these challenges, we propose *DuetInfer*, a modality-aware serving framework that enhances single-GPU serving of VLMs. *DuetInfer* enables parallel computation through preprocessing parallelism and decoupled execution of vision encoding and language modeling. However, such a design necessitates solving two ensuing problems: redundant resource allocation and impaired request fairness. To overcome the redundancy, *DuetInfer* incorporates a shared-centric memory manager to eliminate redundant weight storage and support

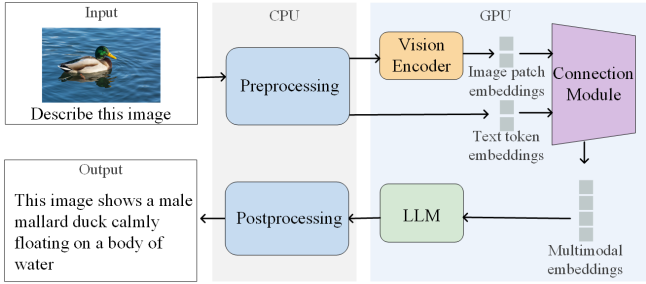


Fig. 2. The VLM inference pipeline

efficient encoder cache reuse, and employs a fairness-aware scheduler to mitigate multimodal delays while preserving the responsiveness of text-only requests.

Overall, we make the following contributions:

- We design an execution scheme that identifies request modality upon arrival, enabling concurrent preprocessing of text-only and multimodal requests. Furthermore, by decoupling vision encoding from language modeling and scheduling them asynchronously, *DuetInfer* alleviates head-of-line blocking and reduces overall latency.
- We design a sharing-centric memory management mechanism that minimizes redundancy and enables encoder cache sharing across requests.
- We present a fairness-aware scheduling policy that reduces disproportionate delays for multimodal requests without penalizing text-only tasks.
- We integrate *DuetInfer* into the vLLM serving engine. Under mixed workloads, *DuetInfer* reduces P99 TTFT by up to 33.7% and end-to-end latency by up to 20%.

II. BACKGROUND AND MOTIVATION

A. Vision-Language Model

VLMs extend conventional LLMs by incorporating multimodal processing modules, enabling the understanding and generation of diverse inputs such as text, audio, images, and videos [13]. This paper primarily focuses on the image modality in conjunction with text inputs as illustrated in Fig. 2. The VLM inference pipeline generally comprises three primary stages: Preprocessing, Generation, and Postprocessing.

Preprocessing and Postprocessing are generally performed on the CPU. During preprocessing, text input is tokenized, whereas image input is converted to tensors of pixel values, cropped, and transformed before being passed into the model [14]. Postprocessing decodes the tokens generated by the LLM into text.

The generation stage is typically executed and accelerated by the GPU, consisting of a visual encoder, a connection module, and the LLM backend. Text tokens are first mapped to embeddings via an embedding layer of LLM. Meanwhile, image patches are processed by a visual encoder—typically employing a ViT backbone [14]—that projects each patch into an embedding with positional encoding. The connection module, usually implemented as a linear projection layer, aligns the dimensionality of image embeddings with that of text embeddings. Once aligned, they share the same embedding

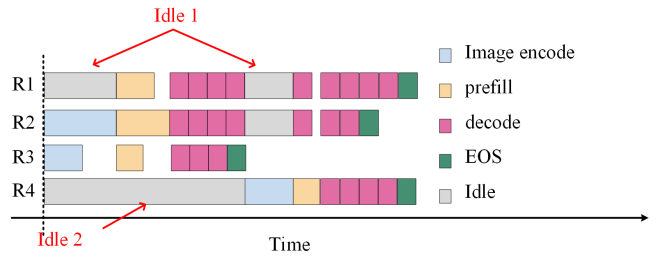


Fig. 3. The GPU timeline of VLM inference

dimension and can be concatenated as input to the LLM. The LLM uses a transformer-based architecture to decode the fused multimodal representations into text [15], [16] through prefilling and decoding.

B. Inference Serving System

Efficient inference serving has become increasingly important for LLMs, as latency directly affects user experience while throughput profoundly impacts operational costs. In recent years, a variety of serving optimization techniques have been proposed, including prefill chunking, prefix caching, continuous batching, and speculative decoding. For example, Orca [17] introduced continuous batching, which dynamically groups multiple requests into a single batch for execution. At each decoding step, the scheduler immediately replaces completed requests with new ones from the waiting queue - without waiting for the entire batch to finish. In addition, well-known serving engines such as vLLM [12] and SGLang [18] allow CPU and GPU to operate on their respective workloads concurrently by decomposing computation tasks and adopting multi-process architectures, thereby improving resource utilization.

With respect to VLMs, the execution pipeline incorporates additional critical stages such as the vision encoder and connection module before reaching the LLM backend. Two latency metrics are particularly critical in evaluating performance: Time-to-First-Token (TTFT) and Time-per-Output-Token (TPOT) [19]. Specifically, TTFT measures the latency from request arrival to the generation of the first token. TTFT covers text tokenization and prefill phase for text-only request, however, for multimodal request, it additionally includes vision preprocessing and vision encoding. As a result, TTFT experiences a significant increase for multimodal request.

C. Motivation

Unfortunately, existing serving systems including vLLM and SGLang treat all requests uniformly, without distinguishing between multimodal and text-only requests. Consequently, multimodal requests cannot be executed efficiently. Compared with text-only requests, multimodal requests involve additional stages [20]–[22]. In the preprocessing stage, text-only requests require only text tokenization, whereas multimodal requests additionally involve vision preprocessing, whose computational complexity is substantially higher. In the generation stage, text-only requests can directly enter text generation (prefill and decode), while multimodal requests must first undergo vision encoding before proceeding to text generation.

As illustrated in Fig. 3, for simplicity of explanation, we assume the batch size is 3—meaning a maximum of 3 requests can be executed concurrently. Suppose at a given moment, 4 requests (R1–R4) arrive simultaneously, where R1 is a text-only request and the rest are multimodal requests. Under the default FCFS scheduling policy adopted by current serving systems, R1, R2, and R3 are admitted to the active batch for execution, while R4 must wait until any one of the requests in the current batch is completed before it can be executed. During this procedure, two notable sources of idle time can be observed: *First*, within a batch, if any request requires vision encoding, all other requests — whether in the prefill or decode phase — must wait until this operation is completed. For example, R1’s prefill phase cannot start until R2 and R3 finish their vision encoding, and likewise, the decoding of R1 and R2 must wait for R4’s vision encoding to complete. *Second*, when the batch is already full, newly arriving requests cannot be scheduled until one of the active requests finishes. For instance, R4 must wait until R3 completes before it can begin execution, which further prolongs its latency.

Based on the above observations, we introduce *DuetInfer*, aiming to address the inefficiencies of VLM inference under mixed workloads on a single GPU. Our work is driven by two key objectives: 1) to reduce the latency and inefficiency of multimodal requests, where vision encoding blocks language modeling within a batch, and newly arrived multimodal requests are prevented from performing vision encoding when the current batch is already full and executing, despite the opportunity for concurrent execution; and (2) to improve fairness for text-only requests, ensuring that lightweight queries are not blocked by heavy multimodal preprocessing or vision encoding.

III. THE DESIGN OF DUETINFER

A. Design Overview

DuetInfer is a parallel serving architecture that separates preprocessing across modalities and decouples vision encoding from language modeling—with the goal of achieving both fairness and efficiency under mixed workloads on a single GPU.

As illustrated in Fig. 4, on the CPU side, *DuetInfer* employs modality-aware preprocessing parallelism: it dispatches text-only and multimodal requests into distinct queues, enabling concurrent execution of preprocessing tasks for different request types. On the GPU side, the architecture further decouples two compute-intensive stages—vision encoding and language modeling—into specialized, dedicated workers such as Language-Model (LM) worker and Vision-Encode (VE) worker. To facilitate efficient resource sharing across these workers, a shared memory manager is integrated to maintain common resources (e.g., model weights and encoder cache), eliminating redundant memory overhead.

Within the LM worker, a fairness-aware scheduler mitigates the inherent delay of multimodal requests. This scheduler expedites the admission of delayed multimodal requests into inference batches while dynamically adjusting resource allocation to prevent starvation of text-only requests—ensuring both fairness across request types and overall execution efficiency.

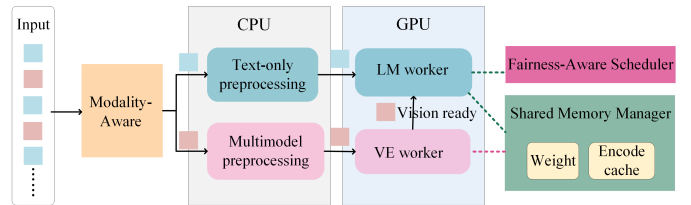


Fig. 4. The system architecture of DuetInfer.

B. Modality-Aware Parallelism

1) **Preprocessing:** Existing serving systems process all requests in a unified queue, which forces lightweight text-only requests to wait behind compute-heavy multimodal preprocessing during the preprocessing phase, thereby inflating their latency. To mitigate this imbalance, we introduce a modality-aware dispatch mechanism that separates preprocessing into two concurrent execution paths for text-only and multimodal requests. Upon arrival, requests are classified and placed into the corresponding queue in terms of modality, each served by a dedicated worker. The workers then perform modality-specific operations—tokenization for text-only requests and image-to-tensor transformations for multimodal inputs. The processed outputs are subsequently merged into a unified request stream for GPU execution. Modality-aware dispatch mechanism makes it possible to leverage multicore CPU parallelism, thereby alleviating the preprocessing bottleneck in the inference of VLMs.

2) **Decoupling:** In the generation stage, existing serving systems adopt continuous batching. However, multimodal requests must wait for the completion of vision encoding before entering language modeling, which can cause blocking for text-only requests within the same batch. To address this issue, we introduce an architectural decoupling of vision encoding from language modeling, decomposing multimodal computation into two specialized components: the VE worker and the LM worker. This design further enables multimodal requests to perform vision encoding ahead of batch admission, thereby reducing unnecessary latency caused by the impact of computationally intensive vision encoding on language modeling.

To enable parallel execution of vision encoding and language modeling, we adopt a process-based architecture where each worker operates independently. However, implementing reliable concurrency directly via Python scripts in frameworks like vLLM presents significant challenges, because Python’s Global Interpreter Lock (GIL) and the inherent limitations of low-level GPU resource management result in suboptimal parallelism, where context isolation prevents true overlap and incurs non-trivial context-switching overhead. To address this challenge, we employ NVIDIA’s Multi-Process Service (MPS) [23], which provides a unified GPU execution environment for multiple processes. MPS eliminates redundant context switches and allows fine-grained sharing of GPU resources, thereby improving utilization of compute resources and memory bandwidth.

Multimodal requests are first dispatched to the VE worker. After vision encoding is completed, the request is forwarded to the scheduler of the LM worker for subsequent processing. In contrast, text-only requests bypass visual encoding entirely and

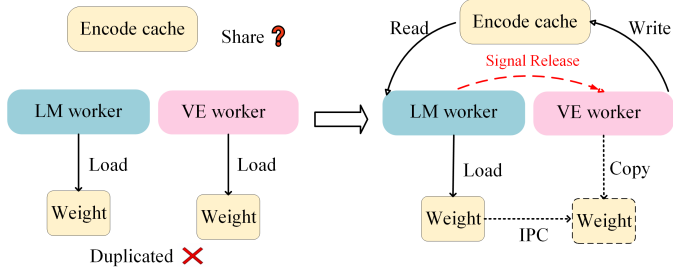


Fig. 5. Sharing-centric memory management.

are directly enqueued in the LM worker, where they undergo prefill and decode operations.

C. Sharing-Centric Memory Management

After the decoupling of the vision encoder and the language model, memory management poses two critical challenges. First, loading the model weights—comprising both the LLM parameters and the vision-encoder parameters—is required by both the VE worker and the LM worker. On a single GPU with constrained memory, maintaining separate copies of the weights in each process can result in severe memory pressure, potentially causing out-of-memory failures. Second, for multimodal requests, the vision embeddings generated by the VE worker must be stored in an encoder cache for subsequent retrieval by the LM worker. Decoupling therefore necessitates efficient cross-process cache sharing.

To address the issue of redundant weight loading, we have implemented the sharing of model weight memory via Inter-Process Communication (IPC) [24], enabling memory-efficient parallel execution for VE and LM worker (Fig. 5). During initialization, the LM worker loads all model weights once and exposes IPC handles to the VE worker. This design allows both workers to directly access a shared weight space without redundant replication, thereby alleviating severe memory overhead that would otherwise arise from duplicating large model weight parameters. Consequently, the risk of out-of-memory errors is significantly reduced under limited GPU memory conditions.

For encode cache management, we have designed a coordinated sharing mechanism between workers. After vision encoding, the VE worker deposits the resulting embeddings into a shared queue. Prior to execution, the LM worker retrieves the required embeddings from this queue to proceed with computations. Since the encode cache is governed by a fixed budget, newly encoded embeddings cannot accumulate indefinitely. Instead, vision encoding must wait until the LM worker frees up memory from completed requests. Once a request finishes processing its embeddings, the LM worker sends a release signal to the VE worker, which then updates the encode cache state before the next scheduling cycle. This coordination mechanism avoids overcommitment of GPU memory capacity and effectively resolves the issue of model runtime failures caused by insufficient memory.

D. Fairness-Aware Scheduler

Following the decoupling of the vision encoding and the language modeling, each worker maintains independent waiting

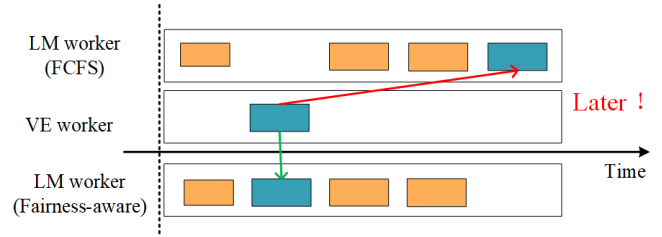


Fig. 6. The waiting queue reordering in the LM worker.

and running queues. Upon completing vision encoding, multimodal requests are inserted into the LM worker’s waiting queue alongside text-only requests. As illustrated in Fig. 6, multimodal requests suffer inherent disadvantages under conventional FCFS scheduling. Although such requests may arrive at the serving system earlier, they must first undergo vision encoding before being enqueued in the LM worker. As a result, this additional delay effectively pushes back the queue position of multimodal requests, placing them behind text-only requests that arrive later. This not only introduces systematic unfairness but also inflates their TTFT.

To mitigate the aforementioned unfairness without starving text-only tasks, we have designed a fairness-aware scheduler that operates on the waiting queue of the LM worker. Newly arriving requests are assigned an initial priority and placed at the corresponding position in the queue based on this priority. To prevent starvation, we incorporate an aging mechanism: when a request’s waiting time exceeds a threshold, defined as the 90th percentile of historical waiting times, its priority is incrementally elevated. This ensures all requests are eventually scheduled, with no task left unserved indefinitely.

The fairness-aware scheduler that jointly considers request arrival time and workload cost estimation. Each priority key is represented as a tuple $(arrival_time, cost_estimate)$, where $arrival_time$ denotes the request’s system-level arrival timestamp, and $cost_estimate$ approximates the computational cost of serving the request. The cost estimation is dynamically computed as Formula 1, where $L_{prefill}$ represents the number of input tokens, and \hat{L}_{decode} is the predicted output length estimated from historical distributions. Using historical distribution data, we developed separate linear regression models for text-only and multimodal requests to predict their respective output lengths. The coefficients α and β are learned from these models and serve as weighting parameters to balance the relative cost between prefill and decode operations. Moving forward, dedicated predictive models will be adopted to enhance the accuracy of output length predictions.

$$cost_estimate = \alpha \cdot L_{prefill} + \beta \cdot \hat{L}_{decode} \quad (1)$$

We also implement the scheduler with a priority queue structured as an AVLtree. Each node consists of a composite priority key and a request list. Nodes are strictly ordered by their priority keys, ensuring deterministic ranking across heterogeneous requests. Within a node, requests sharing identical keys are served in FIFO order, preserving fairness among equivalently prioritized tasks.

TABLE I
MODEL SIZES AND SERVER CONFIGURATIONS

Model	Parameter	GPU	KV Cache
Qwen2-VL-2B	4.15 GB	RTX 4090/24GB	373,216 tokens
Qwen2-VL-7B	15.53 GB	A800 PCIe/80GB	812,528 tokens
Qwen2.5-VL-7B	15.63 GB	A800 PCIe/80GB	812,624 tokens

IV. EVALUATION

A. Experimental Setup

Testbed and Serving Models: We build *DuetInfer* atop vLLM 0.8.2 for efficient multimodal inference on a single GPU. Experiments are conducted on two NVIDIA GPUs as shown in Table I. Qwen2-VL-7B and Qwen2.5-VL-7B are deployed on the A800 for end-to-end performance evaluation, while Qwen2-VL-2B runs on the 4090 for ablation and sensitivity studies. The software stack consists of CUDA 12.4 and PyTorch 2.6.0, ensuring compatibility with modern GPU acceleration libraries. All models were operated with FP16 precision to balance inference accuracy and computational efficiency.

Workloads: To emulate heterogeneous deployment scenarios, we construct mixed workloads using ShareGPT [25] and COCO Captions [26]. ShareGPT provides diverse real-world conversational text and serves as the source of text-only inputs, while COCO Captions contributes multimodal inputs. Fig. 7 illustrates the input length distributions of both datasets. We combine samples from the two sources and apply a weighted random shuffle to generate controlled ratios of text-only and multimodal requests. Since neither of the datasets contains request arrival timestamps, we synthesize request arrival events using a Poisson process with configurable arrival rates—a standard approximation method widely adopted in prior work.

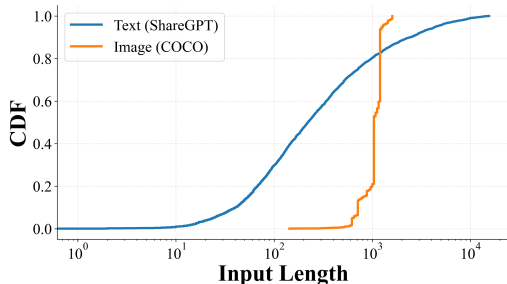


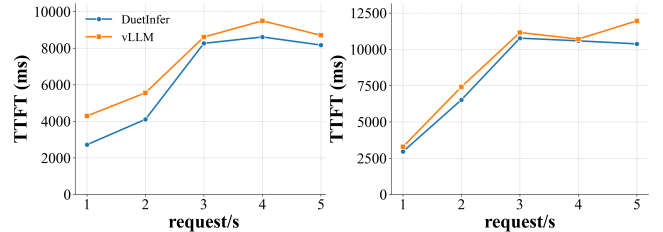
Fig. 7. Input length distributions for image (COCO Captions) and text (ShareGPT) Requests.

Metrics and Baseline: In this work, we primarily focus on end-to-end latency, including TTFT and TPOT. To capture tail performance, we report the P99 values of TTFT and TPOT across all evaluated systems. We use vLLM as the primary baseline to quantify the improvements brought by our proposed scheduling architecture.

B. End-to-End Performance

We systematically evaluate the P99 latency performance of *DuetInfer* against vLLM on Qwen2-VL-7B and Qwen2.5-VL-7B. The experiments are conducted under mixed workloads

consisting of 20% text-only and 80% multimodal requests. As shown in Fig. 8, the P99 TTFT of *DuetInfer* consistently outperforms vLLM as the request rate increases, achieving up to a 33.7% reduction. This performance improvement primarily arises from *DuetInfer*'s parallel architecture design, which enables concurrent preprocessing and decouples vision encoding from language model inference, thereby effectively reducing head-of-line blocking and improving the efficiency of multimodal inference.



(a) Qwen2-VL-7B (b) Qwen2.5-VL-7B

Fig. 8. P99 TTFT for different models. Lower is better.

We report average P99 TPOT measured over request rates of 1–5 requests/s in Table II. *DuetInfer* reduces by up to 80.8% relative to vLLM. This significant improvement is primarily attributed to decoupling visual encoding from language decoding: by removing the head-of-line blocking caused by heavy, bursty visual encoding, the decoder can maintain a steady, uninterrupted GPU decoding stream, which substantially lowers per-token tail latency.

TABLE II
P99 TPOT COMPARISON ON DIFFERENT MODELS

Model	vLLM	<i>DuetInfer</i>
Qwen2-VL-7B	146.6 ms	30.2 ms
Qwen2.5-VL-7B	145.6 ms	28.0 ms

Motivated by the inefficiencies of mixed-modality inference on a single GPU, we evaluate the impact of *DuetInfer* on both tail latency and average latency. As illustrated in Fig. 9, the performance gains extend beyond P99 latency improvements to also boost overall system responsiveness. Specifically, *DuetInfer* reduces the P99 TTFT for multimodal requests by 15.05% relative to vLLM, while achieving comparable or slightly improved TTFT (4.47% reduction) for text-only requests, thereby narrowing the fairness gap between heavyweight multimodal and lightweight text-only workloads. In addition, *DuetInfer* lowers the mean end-to-end latency by 9.65% across different request rates, demonstrating its effectiveness in improving overall latency under mixed workloads.

C. Ablation Study

We conduct an ablation study on Qwen2-VL-2B to systematically evaluate the components of *DuetInfer*, including parallel preprocessing, vision-language decoupling, and the fair scheduler. The result is shown as Fig. 10:

First, parallel preprocessing significantly reduces the P99 TTFT of text-only requests by up to 90%, while also lowering

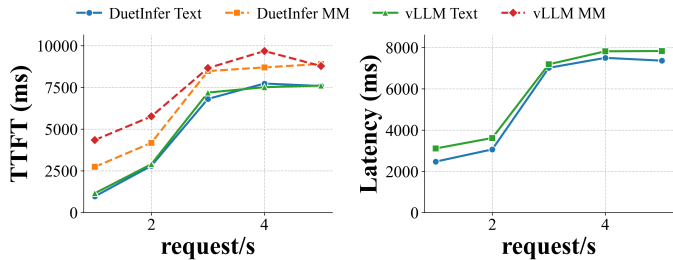


Fig. 9. (a) P99 TTFT for text-only vs. multimodal requests; (b) mean end-to-end latency. Comparison between vLLM and *DuetInfer* under mixed workloads. Lower is better.

the TTFT of multimodal requests by up to 35%. These results show that eliminating the sequential preprocessing bottleneck can mitigate head-of-line blocking caused by vision processing, thereby improving fairness across heterogeneous request types.

Second, we analyze the impact of decoupling vision encoding from language model decoding. The results indicate that this separation initially elevates the TTFT of multimodal requests (described in Section II-C): multimodal requests may enter the queue later than their actual arrival time. After incorporating the fairness-aware scheduler, the multimodal TTFT is reduced by up to 34% without compromising the tail latency of text-only requests, with the improvement being most pronounced under lower request rates.

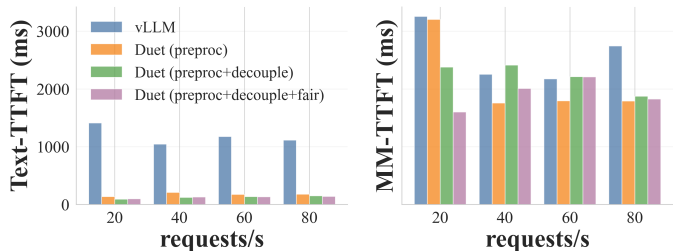


Fig. 10. Ablation study of P99 TTFT across request rates. Left: text. Right: P99 multimodal.

D. Sensitivity Study

We conduct a sensitivity study on Qwen2-VL-2B to examine how the ratio of text-only and multimodal requests impacts P99 TTFT. As shown in Fig. 11, we observe that as the proportion of multimodal requests increases, the reduction in TTFT becomes more pronounced. When the workload consists of 100% multimodal requests, *DuetInfer* achieves a 19.2% reduction in P99 TTFT, demonstrating its effectiveness in scenarios where multimodal requests dominate the workload. In contrast, when text-only requests constitute up to 80% of the workload, there is an increase in TTFT. This observation aligns with our design intent: *DuetInfer* is optimized for multimodal requests by eliminating the blocking effects of expensive visual encoding. When multimodal traffic is low, these benefits disappear but the system still incurs the overhead of the decoupled scheduling logic, causing P99 TTFT to increase for text-only requests.

V. RELATED WORK

Recent years have witnessed growing attention to large model inference optimization research, with specific focus on

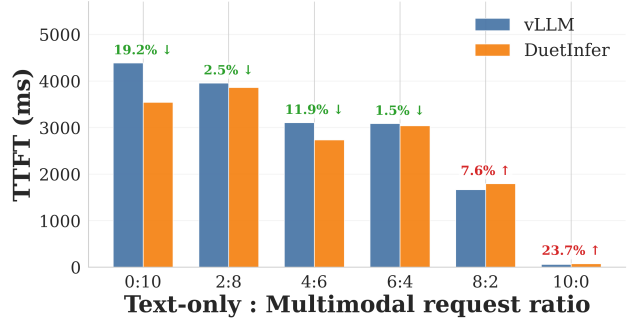


Fig. 11. Sensitivity analysis of P99 TTFT. Each pair of bars corresponds to a workload mix with various request ratios of text-only to multimodal.

two critical components: KV cache and request scheduling.

Orca [17] pioneered iteration-level scheduling, achieving fine-grained resource allocation through dynamic batch reshaping and preemptive execution. Meanwhile, SGLang [18] introduced RadixAttention, which uses a radix tree to enable cross-request KV cache reuse, supporting LRU eviction and cache-aware scheduling. vLLM’s PagedAttention [12] manages attention caches by paging, substantially reducing memory footprint during long-sequence inference and freeing additional GPU memory for concurrent models. vLLM further integrates continuous batching, prefix caching, and quantized inference. ModServe [27] pre-partitions an entire GPU cluster to separately handle vision and language computations. FastServe [28] was the first to introduce token-level preemption, allowing long-sequence decoding tasks to be sliced and preventing any single request from monopolizing resources. FastSwitch [29] decouples prefill and decode stages at the iteration level through dynamic KV cache migration and preemptive context switching. QLLM [30] incorporates request-priority awareness, reordering scheduling to alleviate head-of-line blocking and priority inversion, thus significantly improving response time for high-priority tasks.

In short, these approaches are only applicable to pure-text models and fail to distinguish between multimodal and text-only inputs. In contrast, *DuetInfer* has reconstructed the entire path of the vLLM inference engine, implementing a multimodal-friendly scheduling strategy.

VI. CONCLUSION

We propose *DuetInfer*, a modality-aware serving framework that leverages preprocessing parallelism, decoupled vision–language execution, and fairness-aware scheduling to improve single-GPU serving of VLMs. The experimental results highlight the effectiveness of parallel, fairness-oriented designs for real-world multimodal inference. As future work, we plan to extend *DuetInfer* to multi-GPU cluster environments, thereby enabling scalable coordination between VE and LM components that is well-suited for real-world production deployments.

ACKNOWLEDGEMENT

This work was supported by the Beijing Natural Science Foundation (4212017 and 24L70113) and the State Key Laboratory of Processors, ICT, CAS (CLQ202410). Yuanchao Xu and Keni Qiu are the corresponding authors of this paper.

REFERENCES

- [1] Robert Dale. Gpt-3: What's it good for? *Natural Language Engineering*, pages 113–118, 2021.
- [2] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, 2019.
- [3] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, 2019.
- [4] A Ghosh, A Acharya, S Saha, V Jain, and A Chadha. Exploring the frontier of vision-language models: A survey of current methodologies and future directions. *arXiv preprint arXiv:2404.07214*, 2024.
- [5] Katharine Sanderson. Gpt-4 is here: What scientists think. *Nature*, 2023.
- [6] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [7] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.
- [8] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913, 2017.
- [9] HuggingFace. Image-text-to-text models. https://huggingface.co/models?pipeline_tag=image-text-to-text, 2024. Accessed: 2024-01-01.
- [10] Zongxia Li, Xiyang Wu, Hongyang Du, Fuxiao Liu, Huy Nghiem, and Guangyao Shi. A survey of state of the art large vision language models: Benchmark evaluations and challenges. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 1587–1606, 2025.
- [11] Jiayang Wu, Wensheng Gan, Zefeng Chen, Shicheng Wan, and Philip S Yu. Multimodal large language models: A survey. In *2023 IEEE International Conference on Big Data (BigData)*, pages 2247–2256, 2023.
- [12] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pages 611–626, 2023.
- [13] Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024.
- [14] Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- [15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [16] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- [17] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 521–538, 2022.
- [18] Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37:62557–62583, 2024.
- [19] Amey Agrawal, Anmol Agarwal, Nitin Kedia, Jayashree Mohan, Souvik Kundu, Nipun Kwatra, Ramachandran Ramjee, and Alexey Tumanov. Metron: Holistic performance evaluation framework for llm inference systems. *arXiv e-prints*, pages arXiv-2407, 2024.
- [20] Yuhang Xu, Shengzhong Liu, Dong Zhang, Bingheng Yan, Fan Wu, and Guihai Chen. Nova: Real-time agentic vision-language model serving with adaptive cross-stage parallelization. *arXiv preprint arXiv:2509.21301*, 2025.
- [21] Pavan Kumar Anasosalu Vasu, Fartash Faghri, Chun-Liang Li, Cem Koc, Nate True, Albert Antony, Gokula Santhanam, James Gabriel, Peter Grasch, Oncel Tuzel, et al. Fastvlm: Efficient vision encoding for vision language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 19769–19780, 2025.
- [22] Zhuoyan Xu, Khoi Duc Nguyen, Preeti Mukherjee, Saurabh Bagchi, Somali Chaterji, Yingyu Liang, and Yin Li. Learning to inference adaptively for multimodal large language models. *arXiv preprint arXiv:2503.10905*, 2025.
- [23] NVIDIA. Nvidia mps. <https://docs.nvidia.com/deploy/mps/index.html>, 2022.
- [24] NVIDIA. Inter-process communication. https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvlib_docs/index.html#page/DRIVE_OS_Linux_SDK_Development_Guide/Graphics/nvsci_nvsciipc.html.
- [25] Sharegpt. https://modelscope.cn/datasets/AI-ModelScope/sharegpt_gpt4/summary.
- [26] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.
- [27] Haoran Qiu, Anish Biswas, Zihan Zhao, Jayashree Mohan, Alind Khare, Esha Choukse, Íñigo Goiri, Zeyu Zhang, Haiying Shen, Chetan Bansal, Ramachandran Ramjee, and Rodrigo Fonseca. Modserve: Scalable and resource-efficient large multimodal model serving, 2025.
- [28] Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models, 2024.
- [29] Ao Shen, Zhiyao Li, and Mingyu Gao. Fastswitch: Optimizing context switching efficiency in fairness-aware large language model serving. *arXiv preprint arXiv:2411.18424*, 2024.
- [30] Mohammad Siavashi, Faezeh Keshmiri Dindarloo, Dejan Kostic, and Marco Chiesa. Priority-aware preemptive scheduling for mixed-priority workloads in llm inference. In *Proceedings of the 5th Workshop on Machine Learning and Systems*, pages 132–138, 2025.