MindFusion: Communication-Efficient Mobile-Cloud Orchestration for MLLMs

Xuwang Zeng
School of Computer Science
Sun Yat-sen University
Guangzhou, China
zengxw27@mail2.sysu.edu.cn

Qianyi Huang
School of Computer Science
Sun Yat-sen University
Guangzhou, China
huangqy89@mail.sysu.edu.cn

***Abstract*—Multi-modal large language models (MLLMs) are transforming mobile devices, paving the way for revolutionary applications, such as intelligent personal assistants and innovative human-computer interaction paradigms. However, due to the resource constraints of mobile devices, existing approaches upload raw user data (e.g., images, videos) to remote servers for processing, raising critical privacy concerns, as multimedia data can inadvertently expose sensitive information, including users' identities (e.g., facial features) and personal details (e.g., location, residential address). To address this, we propose *MindFusion*, a mobile-cloud cooperative paradigm that executes MLLM tasks while confining raw user data to local devices. *MindFusion* partitions MLLMs into mobile-resident frontend modules (processing raw inputs) and cloud-hosted backend modules (generating responses), transmitting only intermediate features to cloud servers. To mitigate communication overhead during collaborative inference, *MindFusion* introduces a decoding shortcut to eliminate per-token round-trip interactions and a representation compression module to reduce the traffic volume of intermediate results. We implement *MindFusion* on diverse hardware platforms. Evaluation results show that, compared to on-device inference, *MindFusion* reduces first-token latency by 87% and achieves up to $25\times$ speedup in decoding. *MindFusion* reduces network overhead by 70% without degrading model accuracy. Experimental results show that *MindFusion* presents a practical MLLM inference framework for mobile-cloud orchestration.**

***Index Terms*—Multi-modal Large Language Models, Mobile-Cloud Collaboration, Collaborative Inference**

I. Introduction

Multi-modal large language models (MLLMs) have seen rapid adoption on mobile devices, offering advanced image and video analysis features, like real-time text translation from images, object and scene recognition (e.g., identifying landmarks or products). However, running these models entirely on-device faces significant hardware bottlenecks that severely degrade user experience, such as limited memory and insufficient computational capacity. A common strategy for addressing these resource constraints is to partition the model into two components: a lightweight on-device module and a cloud-based module. The on-device module handles the processing of raw inputs. Meanwhile, the cloud-based module performs computationally intensive tasks that benefit from greater processing power and scalability. This distributed architecture helps leverage the capabilities of large-scale AI models on mobile devices.

However, this approach faces significant challenges regarding communication overhead. On one hand, as MLLMs operate in an autoregressive manner, each generated token (e.g., text response) requires sequential computation across distributed components. To predict the next token y_t, the mobile device processes raw inputs (e.g., text prompts or images) through its local front-end layers, generating intermediate activations. These activations are transmitted to the cloud server, which executes back-end layers to compute the probability distribution for y_t. After that, the cloud server returns y_t to the mobile device. The mobile device cannot proceed to y_{t+1} until it receives y_t. This token-level synchronization transforms the computational benefits of cloud offloading into a latency penalty, where frequent round-trip communications dominate end-to-end delay. On the other hand, multi-modal inputs, especially high-resolution images or videos, are encoded into high-dimensional embeddings (e.g., 4096-D vectors per image patch). Transmitting these uncompressed intermediate features consumes excessive bandwidth.

It is not trivial to address this communication overhead. Prior studies have explored enhancing model parallelism by predicting multiple tokens simultaneously (e.g., generating 4 tokens at a time) [1]. While this approach reduces the frequency of round-trip communication between the mobile device and the cloud server, it does not fundamentally resolve the sequential dependencies in autoregressive decoding. To reduce traffic size, typical approaches [2] [3] keep only a subset of "important" features and drop the rest "unimportant" ones. However, the significance of these features can change dynamically during the autoregressive decoding process, as the generation of later tokens may alter the contextual importance of earlier tokens, potentially degrading model accuracy.

In this paper, we present *MindFusion*, a communication-efficient mobile-cloud orchestration paradigm for MLLMs. To reduce the communication overhead, we propose a decoding shortcut on the cloud server. Instead of sending each predicted token back to the mobile device for further decoding, the decoding shortcut projects the predicted token into the intermediate feature space, eliminating the per-token round-trip communication between the mobile device and cloud server.

To reduce the traffic volume of intermediate features, based on the attention scheme during the prefill state, we observe that later tokens in the input query often carry more semantic weight. Therefore, we apply a higher compression ratio to early tokens, while preserving the fidelity of later tokens by avoiding compression.

We implement *MindFusion* based on the popular MLLM, LLaVA-v1.5-7B, using the ggml library [4]. We evaluate *MindFusion* across various hardware platforms and under different network conditions, including both LAN and Internet environments. Evaluation results show that *MindFusion* significantly improves inference efficiency. Compared to on-device deployment, *MindFusion* reduces the first-token latency by 87% and increases the decoding throughput by $25\times$. Furthermore, instead of directly transmitting intermediate features, *MindFusion* reduces network traffic by over 70%, while maintaining model accuracy.

Our main contributions are as follows:

- We design two key modules: a decoding shortcut that eliminates per-token round-trip communication between mobile and cloud during autoregressive decoding, and a representation compression module that significantly reduces the volume of intermediate features.
- We implement *MindFusion* on diverse hardware platforms. Evaluation results show that *MindFusion* can significantly enhance model throughput while preserving accuracy. Moreover, the performance of *MindFusion* is comparable to cloud-based deployment.

II. Background and motivation

A. Multimodal large language model

Multi-modal large language models (MLLMs) integrate heterogeneous modalities (e.g., image, audio, video) via transformer-based architectures and cross-modal attention. The MLLM pipeline generally comprises three stages: **1) Modality-Specific Encoding:** Raw inputs are processed by pretrained encoders (e.g., LLaMA tokenizer, ViT) into semantic embeddings. **2) Cross-Modal Fusion:** A shared transformer backbone interleaves self- and cross-attention to align features, allowing visual features to guide text generation or textual inputs to focus on salient visual regions. **3) Autoregressive Decoding:** The fused representations are passed to a decoder to generate responses conditioned on the context.

Decoding is inherently sequential; each token y_t causally depends on all prior tokens $y_0, \ldots, y_{t-1}$. This is enforced by self-attention and maintained via a dynamically updated key-value (KV) cache. This creates a strict ordering constraint: y_t cannot be computed until y_{t-1} is finalized and its KV cache entries are updated.

B. Mobile-cloud Cooperative MLLM

Running MLLMs locally reduces cloud dependency but faces hardware bottlenecks [5]. Limited mobile memory (e.g., 8–12GB) struggles with 7B models and their KV caches, while low compute capacity results in prohibitive latency (e.g., > 30s for the first token) and low throughput (4–6 tokens/s).

A common solution partitions the MLLM: local frontend layers (encoders and shallow transformer blocks) handle raw data, while backend layers (deep blocks and the decoder) are offloaded to the cloud. This leverages both on-device preprocessing and cloud scalability, yet faces high communication overhead:

1) Sequential Dependency: In the distributed setting, since local frontend layers must be involved in each iteration of the autoregressive loop, each token y_t requires a mobile-cloud round-trip. While an RTX4090 GPU generates a token in $\sim$10 ms, typical Internet RTT (20–200 ms) dominates the end-to-end delay, transforming the computational speedup of offloading into a latency penalty.

2) High Transmission Volume: During the initial prefill stage, multi-modal inputs are encoded into high-dimensional embeddings (e.g., 4096-D vectors). Transmitting uncompressed features for a typical query (e.g., 50 tokens and a 1443×1080 image) can reach 10MB in LLaVA-v1.5-7B, quickly saturating mobile bandwidth and increasing energy consumption.

III. *MindFusion* Overview

In this section, we present a high-level overview of the system architecture. The design of *MindFusion* is driven by a core objective: communication-efficient mobile-edge interaction. We want to minimize the communication overhead between the mobile device and the cloud server. On one hand, *MindFusion* needs to reduce the number of round-trip communications between mobile devices and cloud servers, as the round-trip communication will increase the end-to-end latency; On the other hand, *MindFusion* should shrink the volume of transmitted data (e.g., activations, intermediate features) without compromising model accuracy.

There are two main components in *MindFusion*: *Representation Compression* and *Decoding Shortcut*. Figure 1 shows the framework design and workflow of *MindFusion*.

First, the mobile device processes the image and text input to extract feature embeddings (❶ ❷). With these feature embeddings, the device executes the initial layers of the original MLLM architecture locally. It then employs the encoder in the *representation compression* module to compress the resulting intermediate activation values before transmitting them to the cloud server (❸).

After receiving the intermediate activation values, the cloud server utilizes the decoder of the *representation compression* module to reconstruct the compressed data (❹). It then proceeds to execute the remaining layers of the original MLLM architecture (❺). Upon generating the prediction for a new token, the cloud employs the *decoding shortcut* module to encode and project this fresh token into the MLLM feature space (❻). This projected representation is then fed back into the subsequent layers of the model on the cloud, continuing the autoregressive decoding process iteratively until either a sequence terminator is produced or the output length reaches its predefined maximum. Finally, the cloud returns the complete inference result to the edge device (❼).

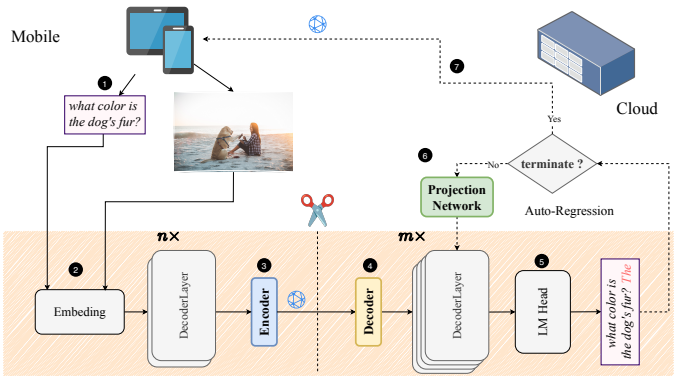

Fig. 1: The system overview of *MindFusion*.

Note that running partial decoding layers on-device helps compute global attention across modalities, which better guides the representation compression module in deciding which features to preserve and which to compress.

IV. *MindFusion* Design

In this section, we present the design details of the two core components in *MindFusion* aimed at minimizing communication overhead.

A. Representation Compression

In mobile-cloud collaborative inference, transmitting uncompressed intermediate features from the mobile device to the cloud consumes excessive bandwidth. This high traffic volume primarily stems from the modality alignment process. For instance, visual encoders expand image embeddings (e.g., from 1024D in CLIP to 4096D in LLaVA-v1.5-7B), which inherently introduces significant redundant information.

Furthermore, due to the causal attention mechanism in autoregressive models, later tokens integrate information from preceding ones. In standard MLLM prompts, user queries typically follow image data. Consequently, text features encapsulate the global context necessary for reasoning, while earlier image features contain redundancy that can be heavily compressed without severe accuracy degradation. Therefore, *MindFusion* prioritizes the compression of intermediate image features while leaving text features uncompressed to preserve fidelity.

We design a symmetric encoder-decoder representation compression network. The mobile-resident encoder compresses the image activations to $1/8$ of their original dimension using two fully connected (FC) layers. The cloud-hosted decoder reconstructs the features back to their original dimension using a mirrored architecture. Formally, let z_i be the activation value of the i-th input token, β be the set of position indices for image tokens, and ω represent the encoder-decoder network. The processed activation $h(z_i)$ is defined as:

$$h(z_i) = \begin{cases} \omega(z_i) & \text{if } i \in \beta \\ z_i & \text{otherwise} \end{cases} \tag{1}$$

Type	Device	Backend
Mobile	iPhone 15 Pro (8G), Redmi K70 Pro (16G)	Metal, CPU
Edge	Macbook Air M2 (16GB), Mac mini M4 (16GB)	Metal
Cloud	Nvidia A100 (40G), RTX4090 (24G)	CUDA

TABLE I: Specifications of the devices used in the evaluation.

Let the partitioned MLLM be $p_\delta = p_{\theta_2} \circ h \circ p_{\theta_1}$, where p_{θ_1} and p_{θ_2} denote the mobile and cloud modules, respectively. Keeping the original MLLM parameters frozen, we train the codec network using the following loss:

$$\mathcal{L}_{\text{comp}} = -\frac{1}{T}\sum_{t=1}^{T} \log p_\delta(y_t|\hat{y}_{<t}, x, \beta) \tag{2}$$

B. Decoding Shortcut

Sequential dependencies in autoregressive decoding force token-by-token round-trip communication, bottlenecking collaborative inference. However, once the pre-filling stage is completed, the cloud server already retains the intermediate features and KV cache necessary for subsequent steps.

To eliminate per-token mobile-cloud interactions, *MindFusion* introduces a Decoding Shortcut on the cloud server. This mechanism allows the cloud-hosted backend to bypass the mobile-resident frontend during the decoding phase. Specifically, for each newly predicted token, we utilize the original MLLM tokenizer and embedding layer on the cloud to obtain its initial representation. We then introduce a Projection Network to map this embedding directly into the input feature space of the cloud's first decoder layer. The Projection Network consists of four FC layers: the first maintains the dimension, the second expands it by $4\times$, and the subsequent two layers symmetrically reduce it back to the original size.

With the decoding shortcut υ, the mobile device only participates in the pre-filling stage, and the autoregressive generation is strictly confined to the cloud. The loss function for training the projection network is defined as:

$$\mathcal{L}_{\text{dcut}} = -\frac{1}{T}\left(\log p_\theta(y_1|x) + \sum_{t=2}^{T} \log p_{\theta,\upsilon}(y_t|\hat{y}_{<t}, x)\right) \tag{3}$$

For the first predicted token, the shortcut is not applied, retaining the original model's loss term. For subsequent tokens, the generation relies on the projection network $p_{\theta,\upsilon}$.

V. Evaluation

To evaluate the performance of *MindFusion*, we conduct extensive experiments on image-based multimodal tasks. The evaluation focuses on two key aspects: latency and throughput, the balance between accuracy and communication overhead.

A. Experimental Setup

Table I lists the devices used in our evaluation. For mobile devices, we conducted tests on both iOS and Android platforms, including the iPhone 15 Pro and the Redmi K70 Pro. For the cloud server, experiments were performed using two different GPUs: the Nvidia A100 and the RTX 4090. The two different GPUs are installed on the same server, which is

Strategy	Device	Data transmission delay	Time to first token	Generation speed	Pre-filling speed
Mobile	iPhone 15 Pro		32s	5.38 token/s	30.08 token/s
	Redmi K70 Pro (CPU)		94.9s	4.53 token/s	6.52 token/s
Edge	Apple M2		5.6s	17.41 token/s	135.56 token/s
	Apple M4		3.8s	21.42 token/s	193.75 token/s
Cloud	Nvidia A100		0.26s	129.40 token/s	2315.05 token/s
	Nvidia RTX4090		0.19s	140.75 token/s	3099.03 token/s
MindFusion (Mobile+Edge)	iPhone 15 Pro + M2	LAN (91ms)	5.91s	16.22 token/s	103.21 token/s
	iPhone 15 Pro + M4	LAN (58ms)	4.03s	19.81 token/s	122.83 token/s
	Redmi K70 Pro + M2	LAN (116ms)	11.37s	16.25 token/s	54.08 token/s
	Redmi K70 Pro + M4	LAN (56ms)	10.23s	19.84 token/s	60.12 token/s
MindFusion (Mobile+Cloud)	iPhone 15 Pro + A100	Internet (3.38s)	4.07s	128.53 token/s	151.10 token/s
	iPhone 15 Pro + RTX4090	Internet (3.43s)	4.03s	140.15 token/s	152.60 token/s
	Redmi K70 Pro + A100	Internet (3.41s)	11.29s	128.27 token/s	54.47 token/s
	Redmi K70 Pro + RTX4090	Internet (3.53s)	11.23s	140.13 token/s	54.76 token/s

TABLE II: Delay and throughput performance of *MindFusion*

equipped with an Intel(R) Xeon(R) Gold 6240 CPU and 202 GB of memory. The mobile devices connect to the Internet via the campus WiFi network, whereas the cloud server is deployed within an intranet environment. Requests from the mobile device are forwarded via a transit server (equipped with a public IP address) to the cloud server in the intranet. After receiving a message from the mobile device, the cloud server immediately replies with a short acknowledgment message. We define network latency as the time elapsed from when the mobile device starts sending the message to when it receives this acknowledgment.

While *MindFusion* is primarily designed as a mobile-cloud collaborative inference framework, it can also be extended to a mobile-edge collaborative setup. To explore this extension, we included two edge devices in the evaluation: the Apple MacBook Air M2 and the Mac mini M4. Both the mobile and edge devices are connected to the same campus Wi-Fi access point (802.11ac). By comparing the "mobile + cloud" setting with the "mobile + edge" setting, we can assess the impact of network delay on model latency.

In the subsequent experiments, unless otherwise specified, we use LLaVA-v1.5-7B as the base MLLM. The representation compression module reduces the feature dimensions to $\frac{1}{8}$ of their original size.

B. Inference Speed and Latency

We evaluated the collaborative reasoning performance between mobile devices and cloud servers, as well as between mobile devices and edge servers. The detailed results can be found in Table II.

Mobile + Cloud Setting: The network transmission latency is approximately 3.5 seconds, and the first-word latency is reduced from over 30 seconds to around 4 seconds, achieving a reduction of more than 87%. The pre-filling stage exhibits a noticeable slowdown compared to cloud-only inference, which is primarily caused by network transmission delays and the slower execution of the initial layers on the mobile device. In the decoding stage, the model's generation speed reaches 120-140 tokens per second, representing nearly a 25× improvement compared to on-device inference, with no significant degradation compared to cloud-only inference.

Mobile + Edge Setting: The network transmission latency is approximately 100 ms, and the first-word latency is reduced from over 30 seconds to around 6 seconds, achieving a reduction of 80%. During the pre-filling stage, the processing speed experiences only a slight decrease compared to edge-only inference. In the decoding stage, the model's generation speed increases from the original 4-5 tokens per second to 17-20 tokens per second, representing a 3-5× improvement.

Whether on a local network or the Internet, *MindFusion* demonstrates superior performance in model inference, effectively meeting the real-time requirements of multimodal tasks.

C. Communication Overhead vs. Accuracy

In this subsection, we focus on evaluating the communication overhead of *MindFusion*. To enable fair comparison with SOTA baselines, our experiments cover seven widely adopted benchmarks, including GQA [6], MMBench (MMB) [7], MME [8], VizWiz [9], SQA [10], VQA^{V2}(VQA V2) [11], and VQA^{Text}(VQA Text) [12]. To quantify communication overhead, we define the *feature compression ratio* as the ratio of the compressed size (in bytes) of features or activation values to their original size. A lower ratio signifies less communication overhead.

1) Representation compression: We compare three different compression strategies: **text-only**, **image-only**, and **text+image**. The **text-only** strategy compresses only the activation values corresponding to text, while the **image-only** strategy solely compresses the activation values corresponding to images. In contrast, the **text+image** strategy compresses the activation values corresponding to both text and images. Table III presents the prediction accuracy of different compression strategies on the VizWiz dataset. When applying the text+image strategy, the model's accuracy shows a noticeable decline under both 0.25 and 0.125 compression ratios compared to the baseline. Using the text-only strategy does not result in a significant improvement in accuracy compared to the text+image strategy. However, the image-only strategy achieves a substantial accuracy improvement over the other two strategies, with accuracy levels close to the baseline. It verifies our observation that the activation values corresponding to the text are more important than the image, so we

Metric	Baseline	Image & Text		Text		Image	
		0.25	0.125	0.25	0.125	0.25	0.125
Overall	50.07	41.39	46.76	41.2	46.76	**51.07**	50.91
Other	38.07	38.98	**41.29**	38.81	**41.29**	37.09	36.99
Unanswerable	74.47	41.18	55.52	40.8	55.52	80.87	**81.01**
Yes/No	77.7	78.88	78.37	**79.44**	78.37	78.26	76.29
Number	**49.21**	41.11	35.81	41.11	33.81	39.84	37.14

TABLE III: VizWiz VQA Challenge 2024 Evaluation

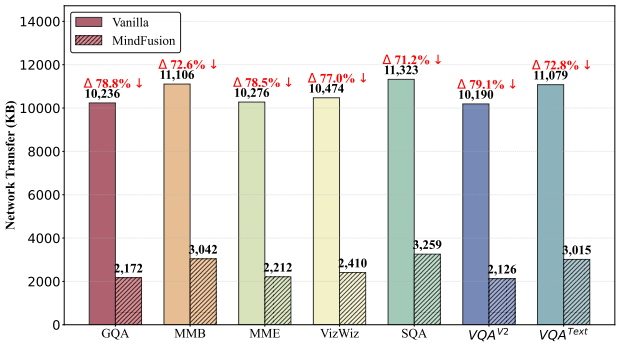

Fig. 2: Average network traffic size of different datasets

can compress the activation values corresponding to the image with high priorities.

In Figure 2, we further evaluated *MindFusion*'s average network traffic volume across different datasets. For each dataset, the left bar represents the original average network traffic volume, while the right bar shows the average network traffic volume after representation compression. Since the question lengths vary across datasets and we compress only the intermediate activation values corresponding to the image, the degree of network traffic reduction varies across datasets. The proportion of savings is higher for datasets with shorter question lengths, such as VQA^{V2}. It is evident that representation compression significantly reduces network traffic, with a reduction of over 70%, effectively lowering network transmission overhead.

We also compared *MindFusion* with several token dropping methods, i.e., LLaVA-PruMerge [13], SparseVLM [14], and MustDrop [3]. As shown in Table IV, *MindFusion* achieves comparable accuracy as the baseline. *MindFusion* maintains higher accuracy compared to existing image feature selection methods, while still attaining a comparable reduction ratio in network traffic.

2) Decoding Shortcut: In this experiment, we evaluate whether the use of the *decoding shortcut* module results in a significant drop in model accuracy. The results are also given in Table IV. Compared with the vanilla model, the decoding shortcut shows minimal impact on accuracy. During inference, large models possess extensive parameters and exhibit strong robustness; even if the shallow network is bypassed, the deeper layers remain capable of compensating for this by effectively capturing the necessary features. Consequently, accuracy undergoes only a minimal decline for all the datasets.

VI. Discussion

In this section, we explore potential directions for further improving *MindFusion*.

Dynamic Compression Ratio: To reduce communication overhead, we apply a uniform compression ratio (e.g., $\frac{1}{8}$) to the activations corresponding to image patches. In modalities such as video, there is significant redundancy along the temporal dimension, and the importance of different video frames varies. A mixed-level codec network, tailored to the characteristics of such modalities, could further optimize bandwidth usage. For instance, key frames (I-frame) in the video could be compressed using a $\frac{1}{4}$ codec network, while predicted frames (P-frame) might be compressed using a $\frac{1}{8}$ codec network.

Multi-Turn Conversation Support: With the decoding shortcut, autoregressive generation is confined to the cloud, meaning the mobile device's local KV cache is not updated with the newly generated tokens. However, complex KV-cache synchronization between the cloud and the mobile device is completely unnecessary for multi-turn conversations. Instead, *MindFusion* can seamlessly handle multi-turn dialogues by simply appending the cloud's response to the conversation history. In the subsequent turn, the mobile device performs a new pre-filling stage over the updated context window. Since the mobile device must encode the newly arrived user inputs regardless, this approach maintains continuous interactions without introducing any synchronization overhead.

VII. Related Work

Split computing: Collaborative reasoning enhances LLM efficiency by partitioning model execution across multiple devices. Recent frameworks [16] leverage heterogeneous computing resources by decoupling inference stages—assigning compute-intensive pre-filling to powerful servers and decoding to edge devices. While traditional partitioning strategies include model and pipeline parallelism, techniques like early exit [17] [18] prove less effective in this distributed setting. This is because collaborative decoding speed is primarily bottlenecked by network latency, rendering the computational savings from early exit negligible.

On-device multimodal large model: Driven by the robust Transformer [19] architecture, Large Language Models (LLMs) such as LLaMA [20] and GPT [21] have revolutionized NLP. This architectural success has rapidly extended to other domains, e.g., Vision Transformers (ViT [22]). Building on these foundations, MLLM integrate diverse modalities via advanced feature extraction and fusion techniques [22].

Deploying MLLM on resource-constrained edge devices necessitates rigorous optimization. To adapt to the limitations of heterogeneous processors, current research focuses on model compression techniques, primarily quantization and pruning [23], which significantly reduce parameter size and computational overhead. Furthermore, optimizing the storage and access patterns of the KV cache [24] during the decoding stage has proven effective in minimizing memory usage and inference latency.

Method[2]	GQA	MMB	MME	VizWiz	SQA	VQA^{V2}	VQA^{Text}	Compression ratio[1]
Vanilla	61.9	64.7	1862	50	69.5	78.5	58.2	100%
ToMe [2]	48.6	43.7	1138		50	57.1	45.3	
FastV [15]	46.1	48	1256		51.1	55	47.8	
LLaVA-PruMerge [13]	48.8	47.4	1201	49.7	50.9	56.2	46.1	11.1%
SparseVLM [14]	52.7	56.2	1505	50.1	62.2	68.2	51.8	
MustDrop [3]	53.1	60	1612	51.2	63.4	69.3	54.2	
MindFusion (with D)	**61.93**	**64.56**	**1864**	49.93	**69.51**	**78.53**	**57.92**	100%
MindFusion (with R)	61.45	62.92	1809	**50.88**	68.07	78.0	56.92	12.5%
MindFusion (with R + D)	61.42	62.94	1809	50.83	68.07	77.96	56.67	12.5%

[1] To enable fair comparison with existing studies, this ratio refers to the compression ratio of image features.
[2] R: Representation Compression; D: Decoding Shortcut.

TABLE IV: Ablation study for representation compression and decoding shortcut

VIII. Conclusion

In this paper, we present *MindFusion*, a communication-efficient mobile-cloud orchestration paradigm. We implement *MindFusion* on diverse hardware platforms. Evaluation results show that, compared to on-device inference, *MindFusion* reduces first-token latency by 87% and achieves up to $25\times$ speedup in decoding. The decoding throughput is comparable to cloud-based deployments. *MindFusion* reduces network overhead by 70% without degrading model accuracy. Experimental results show that *MindFusion* presents a practical MLLM inference framework for mobile-cloud orchestration.

Acknowledgment

This work was supported in part by National Natural Science Foundation of China under Grant 62472452; in part by Guangdong Basic and Applied Basic Research Foundation (No. 2025A1515010262). Q. Huang is the corresponding author.

References

[1] F. Gloeckle, B. Y. Idrissi, B. Rozière, D. Lopez-Paz, and G. Synnaeve, "Better & faster large language models via multi-token prediction," *arXiv preprint arXiv:2404.19737*, 2024.

[2] D. Bolya, C.-Y. Fu, X. Dai, P. Zhang, C. Feichtenhofer, and J. Hoffman, "Token merging: Your vit but faster," *arXiv preprint arXiv:2210.09461*, 2022.

[3] T. Liu, L. Shi, R. Hong, Y. Hu, Q. Yin, and L. Zhang, "Multi-stage vision token dropping: Towards efficient multimodal large language model," *arXiv preprint arXiv:2411.10803*, 2024.

[4] ggml-org, "ggml," 2025. [Online]. Available: https://github.com/ggml-org/ggml

[5] R. Yi, X. Li, Z. Lu, H. Zhang, D. Xu, L. Yang, W. Xie, C. Wang, X. Liu, and M. Xu, "mllm: Fast and lightweight multimodal llm inference engine for mobile and edge devices," 2023. [Online]. Available: https://github.com/UbiquitousLearning/mllm

[6] D. A. Hudson and C. D. Manning, "Gqa: A new dataset for real-world visual reasoning and compositional question answering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6700–6709.

[7] Y. Liu, H. Duan, Y. Zhang, B. Li, S. Zhang, W. Zhao, Y. Yuan, J. Wang, C. He, Z. Liu *et al.*, "Mmbench: Is your multi-modal model an all-around player?" in *European Conference on Computer Vision*. Springer, 2024, pp. 216–233.

[8] C. Fu, P. Chen, Y. Shen, Y. Qin, M. Zhang, X. Lin, J. Yang, X. Zheng, K. Li, X. Sun *et al.*, "Mme: A comprehensive evaluation benchmark for multimodal large language models," in *Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025.

[9] J. P. Bigham, C. Jayant, H. Ji, G. Little, A. Miller, R. C. Miller, R. Miller, A. Tatarowicz, B. White, S. White *et al.*, "Vizwiz: Nearly real-time answers to visual questions," in *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*, 2010, pp. 333–342.

[10] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan, "Learn to explain: Multimodal reasoning via thought chains for science question answering," *Advances in Neural Information Processing Systems*, vol. 35, pp. 2507–2521, 2022.

[11] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. 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*, 2017, pp. 6904–6913.

[12] A. Singh, V. Natarajan, M. Shah, Y. Jiang, X. Chen, D. Batra, D. Parikh, and M. Rohrbach, "Towards vqa models that can read," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8317–8326.

[13] Y. Shang, M. Cai, B. Xu, Y. J. Lee, and Y. Yan, "Llava-prumerge: Adaptive token reduction for efficient large multimodal models," *arXiv preprint arXiv:2403.15388*, 2024.

[14] Y. Zhang, C.-K. Fan, J. Ma, W. Zheng, T. Huang, K. Cheng, D. Gudovskiy, T. Okuno, Y. Nakata, K. Keutzer *et al.*, "Sparsevlm: Visual token sparsification for efficient vision-language model inference," *arXiv preprint arXiv:2410.04417*, 2024.

[15] L. Chen, H. Zhao, T. Liu, S. Bai, J. Lin, C. Zhou, and B. Chang, "An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models," in *European Conference on Computer Vision*. Springer, 2024, pp. 19–35.

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

[17] E. Demir, "Early-exit convolutional neural networks," Master's thesis, Middle East Technical University, 2019.

[18] T. Schuster, A. Fisch, J. Gupta, M. Dehghani, D. Bahri, V. Tran, Y. Tay, and D. Metzler, "Confident adaptive language modeling," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 456–17 472, 2022.

[19] 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.

[20] 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.

[21] OpenAI, "Gpt-4," 2024. [Online]. Available: https://openai.com/index/gpt-4-research/

[22] 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.

[23] R. Yi, L. Guo, S. Wei, A. Zhou, S. Wang, and M. Xu, "Edgemoe: Fast on-device inference of moe-based large language models," *arXiv preprint arXiv:2308.14352*, 2023.

[24] 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.