# Distributed VLMs: Efficient Vision-Language Processing through Cloud-Edge Collaboration

Yuyang Li
*Electrical Engineering*
*Columbia University*
New York, United States
yl5339@columbia.edu

Devika Gumaste
*Electrical Engineering*
*Columbia University*
New York, United States
dg3370@columbia.edu

Mehmet Kerem Turkcan
*Civil Engineering*
*Columbia University*
New York, United States
mkt2126@columbia.edu

Javad Ghaderi
*Electrical Engineering*
*Columbia University*
New York, United States
jg3465@columbia.edu

Gil Zussman
*Electrical Engineering*
*Columbia University*
New York, United States
gil.zussman@columbia.edu

Zoran Kostic
*Electrical Engineering*
*Columbia University*
New York, United States
zk2172@columbia.edu

***Abstract*—Vision Language models (VLMs) have transformed Generative AI by enabling systems to interpret and respond to multi-modal data in real-time. While advancements in edge computing have made it possible to deploy smaller Large Language Models (LLMs) on smartphones and laptops, deploying competent VLMs on edge devices remains challenging due to their high computational demands. Furthermore, cloud-only deployments fail to utilize the evolving processing capabilities at the edge and limit responsiveness. This paper introduces a distributed architecture for VLMs that addresses these limitations by partitioning model components between edge devices and central servers. In this setup, vision components run on edge devices for immediate processing, while language generation of the VLM is handled by a centralized server, resulting in up to 33% improvement in throughput over traditional cloud-only solutions. Moreover, our approach enhances the computational efficiency of off-the-shelf VLM models without the need for model compression techniques. This work demonstrates the scalability and efficiency of a hybrid architecture for VLM deployment and contributes to the discussion on how distributed approaches can improve VLM performance.**

***Index Terms*—vision-language models (VLMs), edge computing, distributed computing, inference optimization, edge-cloud collaboration.**

## I. Introduction

Vision Language models (VLMs) [1]–[4] have experienced a rapid surge in development [5], leading to a multitude of practical applications across diverse domains. From autonomous vehicles [6] and robotic navigation to augmented reality and assistive technologies [7], VLMs are enabling machines to understand and generate language from visual data with remarkable accuracy and versatility. However, despite the growing capabilities of VLMs, their deployment in practical real-time systems faces significant challenges. The high computational and memory demands of VLMs often render them unsuitable for resource-constrained environments. For example, LLaVA-13B [1], a popular state-of-the-art model, requires at least 26GB of memory at FP16 precision, which exceeds the capacity of most edge devices. As a result, model compression [8] and distributed computing have emerged as crucial areas of research, which aim to reduce the size and computational demands of the model without sacrificing performance. Various strategies such as knowledge distillation [9], quantization [10] [11], pruning, model partitioning and distributed inference [12] are being explored to enhance VLM deployment at the edge.

While many of these methods have been effectively applied to LLMs, [13]–[16] VLMs pose unique challenges due to their distinct architecture and processing requirements. A key strength of VLM architecture lies in its modular design, which separates the components for visual processing and language understanding, thus providing a pathway for optimizing their deployment. In this paper, we propose a novel hybrid computing approach that distributes the components of VLMs across edge devices and a central server. Figure 1 shows the layout of our architecture, where edge devices handle visual processing, while more computationally intensive language tasks are delegated to the centralized server. This approach introduces parallelization into the VLM architecture, enhancing overall throughput and enabling effective operation across multiple devices.

Our framework allows heterogeneous devices to function flexibly as either servers or edge nodes based on their computational capabilities, creating a distributed setup. Unlike methods that rely on pruning or quantization, our approach accelerates inference without compression, ensuring that the model's performance remains intact. Moreover, our distributed implementation enhances deployment accessibility: users only need to provide a model with pre-trained weights, which our framework adapts for distributed operation through a partitioning process. This architecture not only improves inference speed but also optimizes resource utilization and reduces latency, broadening the scope of VLM applications in practical settings. Through extensive experimentation, we demonstrate

that our approach enhances performance, paving the way for scalable solutions across various domains while accelerating inference for VLMs in distributed environments.

## II. Related Work

### A. Vision Language Models

Recent VLMs have demonstrated strong multi-modal reasoning abilities across tasks like visual question answering, image captioning, and content generation. Several efforts have focused on adapting VLMs for deployment on edge devices. For instance, Gemini [17], has introduced lightweight VLMs called Gemini Nano, featuring 1.8B and 3.25B parameters, specifically optimized for smartphones. VILA [4], introduced by NVIDIA, uses CLIP [18] vision encoder combined with the LLaMA [19] language model, excelling in multi-image analysis and zero-shot tasks. MobileVLM [20] is an early open-source model, with 1B and 3B parameter versions, tailored for resource-constrained environments.

### B. Distributed Inference

Distributed inference enables efficient processing by leveraging the computational power of multiple devices, often involving cloud-edge collaboration and compute offloading strategies. Considerable work has been done in this domain for large language models (LLMs). For example, EdgeShard [13] partitions LLMs into shards which can be distributed across various devices, optimizing for both latency and throughput. LLM-PQ [15] enhances efficiency with adaptive quantization and phase-aware partitioning on heterogeneous GPU clusters. PerLLM [16] introduces a personalized inference scheduling framework designed for diverse LLM services through edge-cloud collaboration. However, similar efforts specifically targeting vision language models (VLMs) have yet to be undertaken. The unique characteristics and requirements of VLMs necessitate further exploration of distributed inference techniques tailored to their architectures and specific use cases.

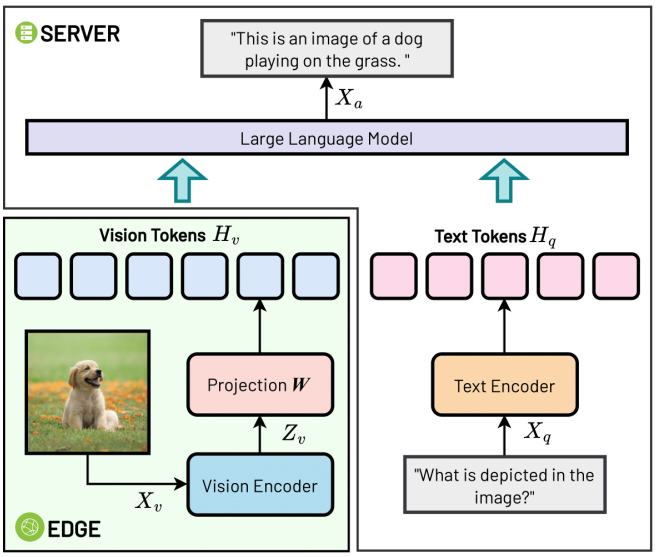



Fig. 1: The edge device identifies the input (image $X_v$ and prompt $X_q$). The prompt $X_q$ is sent to the server, initiating the prompt processing pipeline. Simultaneously, the vision encoder extracts visual features $Z_v$, which are then transformed into tensor $H_v$ via a projection layer. Once ready, $H_v$ is sent to the server where $X_q$ is vectorized into $H_q$. The server then merges the visual and text embeddings into a unified input for the LLM which generates the output $X_a$.

## III. Methodology

The proposed distributed architecture for Vision Language Models (VLMs) is designed to efficiently offload computational tasks between edge devices and a central server, optimizing resource utilization and performance. The system architecture is built around three main components: edge devices, a central server, and a communication system that facilitates interaction between them. The system relies on parallel processing and dynamic flow control to handle real-time vision-language tasks without compromising performance. The flow of data and thread management is illustrated in Figure 2, which provides a detailed overview of the interaction between the edge devices and the central server.

**Algorithm 1** Edge-Side Processing Workflow

**Thread 1: Local Data Processing**
**if** NewImageDetected() **then**
▷ # Retrieve image data and generate unique identifier
$X_q, X_v \leftarrow$ GetData()
$m_x \leftarrow$ GenerateUniqueID()
▷ # Send query text and unique ID to the server
Send$(X_q, m_x) \rightarrow$ Server
▷ # Process visual data
$Z_v \leftarrow$ VisionEncoder$(X_v)$
$H_v \leftarrow$ ProjectionLayer$(Z_v)$
▷ # Send encoded visual features and unique ID to the server
Send$(H_v, m_x) \rightarrow$ Server
**end if**
**Thread 2: Server Status Monitoring**
**if** ReceiveStatus() = PAUSE **then**
Pause Thread-1, await RESUME signal
**else**
Resume Thread-1 processing
**end if**

### A. Edge-Side Processing

The edge device is responsible for encoding images and transmitting the intermediate results to the server. (refer Algorithm-1) Each image has a pre-associated text prompt, which is tracked using a unique input ID. By offloading the entire vision tower to the edge device, we significantly reduce the model size on the server and enhance throughput.

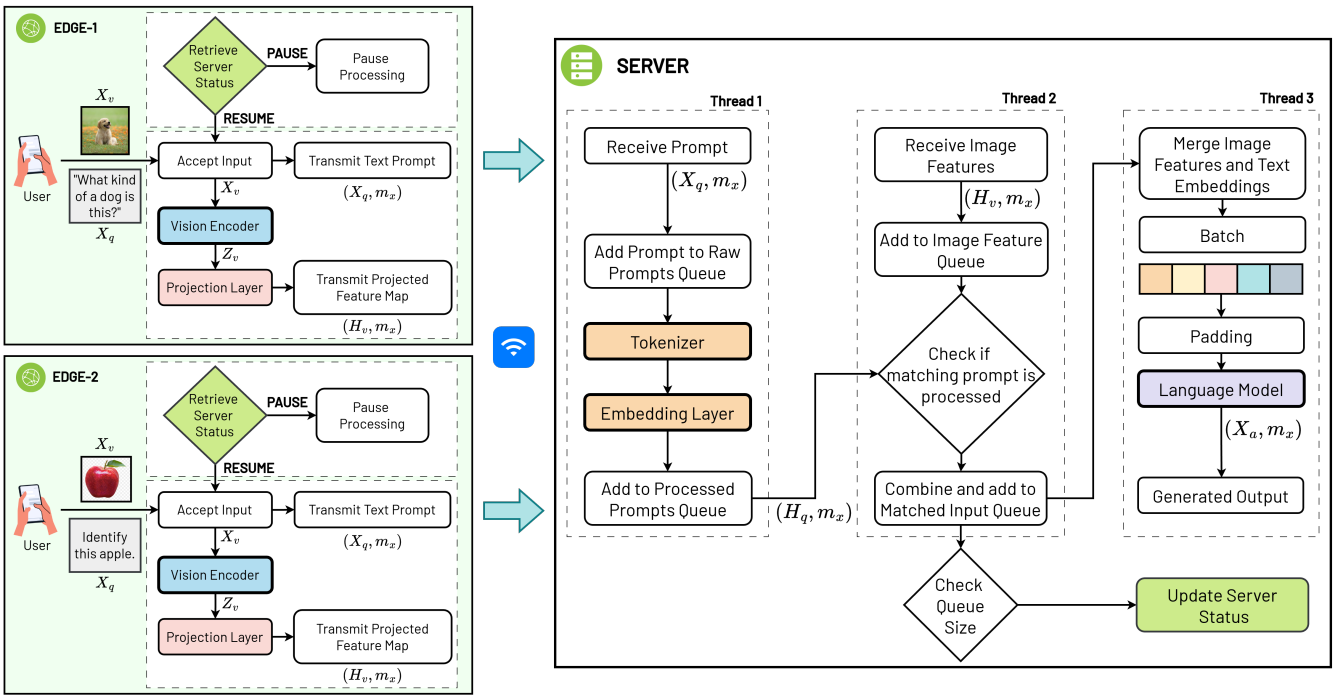



Fig. 2: Distributed VLM Inference: A flowchart depicting the interaction between edge devices and the central server, where each edge device handles image encoding and feature extraction, while the server processes text prompts, matches them with image feature maps, and batch processes the matched inputs. Parallel processing across multiple threads ensures efficient queue management and optimized task handling for high-throughput performance.

*a) Image Processing on Edge:* The edge device operates with a main thread that continuously processes incoming data. Upon receiving a new image, it transmits the associated text prompt to the server to initiate text processing. The image is processed by the vision tower to extract visual features. These features are then passed through a projection layer that converts them into a tensor format compatible with the language model. Once encoded, the image features, along with their corresponding input IDs, are sent to the server, ensuring accurate matching with the text prompts on the server side.

*b) Server Monitoring and Process Control:* On a background thread, the edge device uses a binary flow control mechanism to monitor the server's status. Image processing completely halts upon receiving a "PAUSE" signal and resumes only when clearance is given from the server.

## B. Server-Side Processing

The central server is responsible for managing text prompts, encoded image feature maps, and language model processing (refer Algorithm-2). Each component operates concurrently on different processes to maximize throughput, and the system is designed to support multiple edge devices in parallel, providing scalability for large deployments. The key functions of the server are as follows:

*a) Text and Feature Maps Processing:* Upon receiving a text prompt, with an associated input ID, the server passes the text through an embedding layer to generate dense text embeddings, which are added to the "Processed Prompts" queue. Concurrently, image feature maps are received from the edge device and placed into the "Image Feature Maps" queue. The server implements an efficient matching mechanism using two hash maps to track text embeddings and image features respectively. This hash map-based matching allows $O(n)$ lookup time to identify all corresponding text-image pair sharing the same ID, enabling immediate combination of matched pairs for subsequent LLM processing. Matched pairs are concatenated and enqueued to the "Matched Input" queue, with a threshold-based back-pressure system where the server monitors the "Matched Input" queue occupancy. Upon reaching a configurable threshold, it triggers a PAUSE signal upstream to edge devices, temporarily suspending image submissions until the queue length returns below the threshold, ensuring system stability and preventing memory overflow.

*b) Batch Processing:* Padding is applied to the batch prior to passing it through the VLM's language model for inference. The batch processing thread is designed to operate asynchronously, continuously performing inference on incoming batched inputs while new data is being received and managed in parallel threads. This non-blocking architecture ensures that the batch processing pipeline is not stalled by incoming data or queue management tasks, maximizing GPU utilization. Additionally, by assigning dedicated threads to queue management, matching, and preprocessing, the pipeline minimizes idle times for the LLM and ensures that the language model processes inputs efficiently without delay.

## IV. EXPERIMENTS

The experiments evaluate the performance of the proposed Distributed VLM architecture compared to traditional centralized approaches, with a focus on throughput and scalability.

**Algorithm 2** Server-Side Processing Workflow

**Thread 1: Prompt Processing**
**if** NEWPROMPTRECEIVED() **then**
    ▷ # Generate text embedding for incoming prompt
    $H_q \leftarrow \text{TextEncoder}(X_q)$
    ▷ # Add embedding to the processed prompts queue
    $\text{Enqueue}(H_q, m_x) \rightarrow PP_q$
**end if**
**Thread 2: Feature Processing**
**if** NEWFEATURESRECEIVED() **then**
    ▷ # Add extracted feature vector to the feature queue
    $\text{Enqueue}(H_v, m_x) \rightarrow F_q$
**end if**
**if** $\exists\ m_x \in PP_q$ such that $m_x \in F_q$ **then**
    ▷ # Combine matching feature and prompt embeddings
    $M \leftarrow \text{Combine}(H_v, H_q)$
    ▷ # Add the matched pair to the matched queue
    $\text{Enqueue}(M) \rightarrow M_q$
**end if**
    ▷ # Send PAUSE if queue exceeds threshold
**if** $|M_q| >$ Threshold **then**
    $\text{Send}(\text{PAUSE}) \rightarrow$ Edge
**else**
    $\text{Send}(\text{RESUME}) \rightarrow$ Edge
**end if**
**Thread 3: Batch Processing**
    ▷ # Create a batch from the matched queue
$B \leftarrow \text{Batch}(M_q, \text{MAX\_BATCH\_SIZE})$
▷ # Apply padding to the batch for consistent input size
$B' \leftarrow \text{Padding}(B)$
    ▷ # Run the batch through LLM for inference
$X_a \leftarrow \text{LLM}(B')$
[1]

### A. System setup

For the experiments, we utilized two NVIDIA Jetson [21] devices: 1) *Edge Node:* NVIDIA Jetson Orin Nano 8GB [22], with Edge Side code deployed. 2) *Central Server:* NVIDIA Jetson AGX Orin Developer Kit 64GB [23], with Server Side code deployed. The devices are interconnected via a gigabit network switch, ensuring low network-induced latency and sufficient bandwidth for the system.

### B. Datasets and models

To evaluate the performance of our Distributed VLM system, we profiled three off-the-shelf VLM models with pre-trained weights on a single dataset, comparing the results we obtained from centralized deployment with the distributed deployment strategy we developed. We used the visual riddles dataset [24] to benchmark the VLMs in both centralized and distributed architectures. Our experiments utilized the following models:

- **Moondream-2** [25]: A tiny vision language model of 1.8B parameters using a Phi-1 [26] LLM as the backend. Hereafter referred to as **Moondream2**.
- **LLaVA_QWEN15-4B-chat** [27]: A small 4 billion parameter model that combines the CLIP vision encoder with a Qwen1.5 [28] LLM . Hereafter referred to as **LLaVA-Qwen-4B**.
- **LLaVA-Llama-13B** [1]: A standard LLaVA 1.5 Model based on a CLIP vision encoder and a fine-tuned Llama-2 [19] LLM with 13 billion parameters. Hereafter referred to as **LLaVA-Llama-13B**.

### C. Finding optimal batch sizes

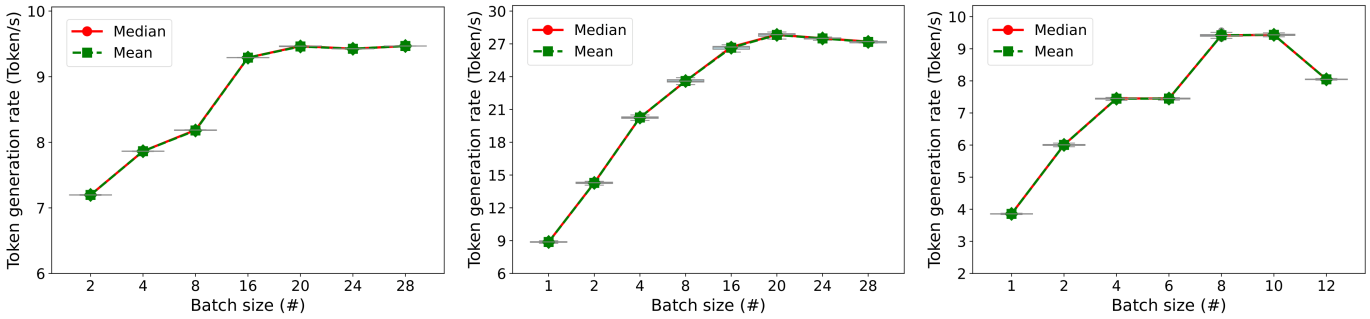
We conducted extensive testing on the central server device, with each experimental run separated by 30 seconds followed by a garbage collection cycle to ensure consistent and reliable measurements. A benchmark of token generation throughput across different VLMs reveals distinct patterns based on model size and architecture, as shown in Figure 3. From the results, we identified the optimal batch sizes for each VLM configuration, with Moondream2 and LLaVA-Qwen-4B both performed best at a batch size of 20, while the larger LLaVA-13B model achieved optimal performance at a batch size of 10. This pattern aligns with theoretical expectations, as smaller models like Moondream2 and LLaVA-Qwen-4B can effectively process larger batches due to their lower memory requirements. In contrast, the more computationally demanding LLaVA-Llama-13B requires a smaller batch size to maintain efficient processing without overwhelming system resources.

### D. Throughput comparison on distributed vs centralized VLM

Using the optimal batch sizes identified in the previous analysis, we conducted a comprehensive throughput comparison between distributed and centralized architectures. To ensure accurate measurements, model throughput was evaluated after an initial warm-up period, ensuring that the LLM on the server always had readily available input for immediate processing. For each model and configuration, we measured the throughput by recording the time required to produce new tokens up to various new token limits and then computed the new tokens generation speed accordingly. These throughput measurements were conducted across three distinct system configurations, with the results shown in Figure 4:

- Centralized approach with a single server device
- Distributed approach with 1 edge and 1 server device
- Distributed approach with 2 edge and 1 server devices

*a) Throughput gain on Moondream2 VLM:* Figure 4 (a) compares token generation throughput between centralized and distributed approaches for the Moondream2 model with a batch size of 20. The distributed architecture demonstrates a clear performance advantage across all token generation




(a) Throughput of Moondream2 (b) Throughput of LLaVA-Qwen-4B (c) Throughput of LLaVA-Llama-13B

Fig. 3: Token generation throughput of different VLMs at various batch sizes, measured on the server.

limits. The performance gain is most significant at lower token limits, peaking at a 33.54% improvement for the new token generation limit set to 5. As the token count increases, the gain gradually decreases, with the system still operating 29% faster than the centralized version at a limit of 15 tokens. Notably, the distributed system with two edge devices did not provide additional throughput gains compared to the single edge device configuration, a trend we observed across other models, as evidenced by the overlapping lines in the graph.

*b) Throughput gain on LLaVA-QWEN-4B VLM:* Figure 4(b) presents a comparison of token generation throughput between centralized and distributed approaches for the LLaVA-QWEN-4B model with a batch size of 20. The throughput gain is most pronounced at lower token generation limits, showing approximately a 15% improvement, and gradually diminishes as the token count increases. Similar to other models, the distributed system with two edge devices performs in a near identical fashion to the single edge device configuration.

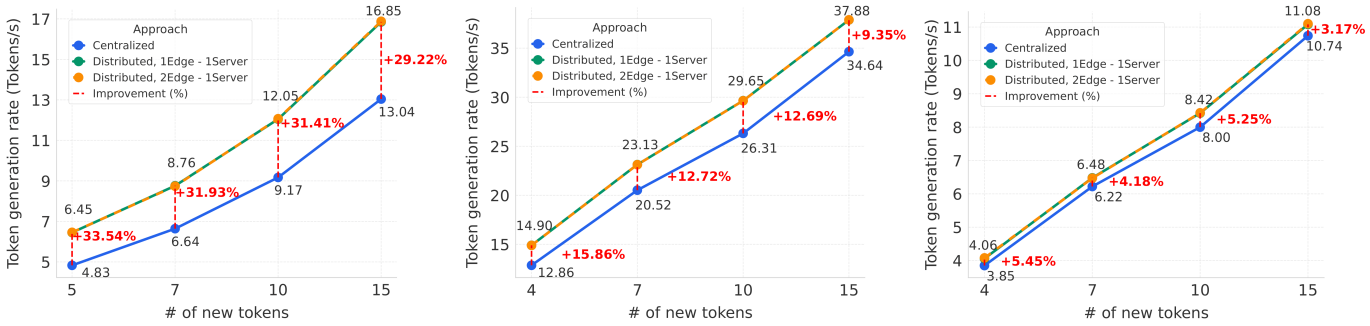
*c) Throughput gain on LLaVA-Llama-13B VLM:* Figure 4(c) presents the throughput comparison between centralized and distributed approaches for the LLaVA-Llama-13B model with a batch size of 10. The distributed architecture shows modest but consistent performance improvements across different new token generation limits. At the lowest token limit of 4, the system achieves a 5.45% throughput gain over the centralized version. This improvement remains relatively stable, gradually decreasing to 3.17% at 15 new tokens. Again, the distributed system with two edge devices performs nearly identically to the single edge device configuration. Although these improvements are less dramatic than those seen in previous models, they still demonstrate that the distributed architecture can provide meaningful performance benefits even for larger language models.

## V. Discussion

Our research on Distributed VLM demonstrates a novel approach to improving the performance of vision language models through a cloud-edge hybrid computing architecture. By strategically distributing model components between edge devices and a central server, we have achieved significant improvements in throughput without compromising model size or relying on compression techniques.

### A. Throughput improvements on different models

The effectiveness of our distributed approach is closely related to the proportion of time spent on vision encoding relative to LLM inference. By profiling generation time for different parts of each vision language model, Table I can be derived to illustrate this point. Upon inspection, the vision encoding time percentage closely mirrors the throughput improvements we achieved across different models. The distributed architecture optimizes system latency by transforming the sequential




(a) Comparison on Moondream-2 (b) Comparison on LLaVA-Qwen-4B (c) Comparison on LLaVA-Llama-13B

Fig. 4: Throughput comparison of Distributed and Centralized VLM at optimal batch sizes, based on token generation rate

TABLE I: Distribution of processing time as a proportion of total processing time for different VLMs under optimal configurations (ideal batch size and new token generation rate).

| VLM Model | Vision Encoding | LLM Inference | Additional Overhead[a] |
|---|---|---|---|
| Moondream2 | 29.1% | 69.4% | 1.5% |
| LLaVA-Qwen-4B | 15.6% | 84.1% | 0.3% |
| LLaVA-Llama-13B | 5.7% | 94.2% | 0.1% |

[a]The additional overhead accounts for system overheads such as data loader and profilers

execution (Vision encoder + LLM inference) into two parallel processing streams. This is achieved by offloading vision encoding to edge devices, enabling concurrent execution with the LLM inference of the previous batch. Through this parallel execution scheme, vision encoding latency is effectively masked by the LLM inference stage of the preceding task, enabling continuous LLM operation without encoder-induced delays. On the other hand, the system's theoretical speedup is thus bounded by the ratio between vision encoding and LLM inference time. Consequently, models with higher LLM overhead like LLaVA-Llama-13B demonstrate limited throughput improvements, while architectures with proportionally heavier vision encoders such as LLaVA-QWEN-4B and Moondream2 achieve more significant efficiency gains.

### *B. Throughput improvements for multiple edge devices*

The consistent throughput across different numbers of edge devices can be attributed to the complete overlap between vision encoding and LLM inference times. As the vision encoding on edge devices is decoupled from the LLM inference on the server, the two processes occurs in a virtually parallel fashion. Adding more edge devices provides no additional benefit once the server's input queue is saturated. This parallel processing effectively masks the vision encoding duration behind the LLM inference time, which remains the primary throughput bottleneck regardless of edge device count. Hence, there's neither improvement nor decline in throughput for each model at each distributed configuration, with one or more edge devices serving encoded images.

### *C. Limitations and Future work*

The Distributed VLM architecture, while effective, faces inherent limitations in its potential for throughput gains. As the system is primarily bottlenecked by LLM inference time, the benefits of offloading the vision encoding task are constrained by the proportion of time it occupies in the overall processing pipeline. This limitation is especially pronounced for larger models where LLM inference dominates the processing time, resulting in diminishing returns for our distributed approach as model size increases. To mitigate this issue, future iterations of our architecture will employ a distributed cluster of devices as a server running the LLM. This will pave the way for even greater efficiency gains in vision-language processing.

## VI. Conclusion

This paper introduces and evaluates the Distributed VLM, an accessible hybrid architecture designed to accelerate Vision Language Model (VLM) inference by distributing tasks between edge devices and servers. The approach offloads vision encoding to the edge, while preserving LLM inference on servers, taking advantage of the modular design of modern VLMs. Our experiments show up to 33% improvements in throughput without sacrificing model performance. Although LLM inference remains a bottleneck, particularly for larger models, this work lays the groundwork for future cloud-edge collaborative systems, demonstrating substantial throughput gains without compromising model capabilities.

## Acknowledgments

This work was supported in part by NSF grant EEC-2133516, NSF grant CNS-2038984 and corresponding support from the Federal Highway Administration (FHA) and MediaTek USA Inc., and ARO grant W911NF2210031.

## References

[1] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," 2023. [Online]. Available: https://arxiv.org/abs/2304.08485

[2] D. Zhu, J. Chen, X. Shen, and X. L. et al., "Minigpt-4: Enhancing vision-language understanding with advanced large language models," 2023. [Online]. Available: https://arxiv.org/abs/2304.10592

[3] J.-B. Alayrac and J. D. et al., "Flamingo: a visual language model for few-shot learning," 2022. [Online]. Available: https://arxiv.org/abs/2204.14198

[4] J. Lin, H. Yin, and W. P. et al., "Vila: On pre-training for visual language models," 2024. [Online]. Available: https://arxiv.org/abs/2312.07533

[5] D. Zhang, Y. Yu, and J. D. et al., "Mm-llms: Recent advances in multimodal large language models," 2024. [Online]. Available: https://arxiv.org/abs/2401.13601

[6] X. Zhou, M. Liu, and E. Y. et al., "Vision language models in autonomous driving: A survey and outlook," 2024. [Online]. Available: https://arxiv.org/abs/2310.14414

[7] C. Li, C. Wong, and S. Z. et al., "Llava-med: Training a large language-and-vision assistant for biomedicine in one day," 2023. [Online]. Available: https://arxiv.org/abs/2306.00890

[8] X. Zhu, J. Li, and Y. L. et al., "A survey on model compression for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2308.07633

[9] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[10] J. Lin, J. Tang, and H. T. et al., "Awq: Activation-aware weight quantization for llm compression and acceleration," 2024. [Online]. Available: https://arxiv.org/abs/2306.00978

[11] A. G. al., "A survey of quantization methods for efficient neural network inference," *ArXiv*, vol. abs/2103.13630, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232352683

[12] J. Stojkovic, C. Zhang, and Íñigo Goiri et al., "Dynamollm: Designing llm inference clusters for performance and energy efficiency," 2024. [Online]. Available: https://arxiv.org/abs/2408.00741

[13] M. Zhang and J. C. et al., "Edgeshard: Efficient llm inference via collaborative edge computing," 2024. [Online]. Available: https://arxiv.org/abs/2405.14371

[14] R. Y. et al., "Edgemoe: Fast on-device inference of moe-based large language models," 2023. [Online]. Available: https://arxiv.org/abs/2308.14352

[15] J. Zhao, B. Wan, and Y. P. et al., "Llm-pq: Serving llm on heterogeneous clusters with phase-aware partition and adaptive quantization," 2024. [Online]. Available: https://arxiv.org/abs/2403.01136

[16] Z. Yang and Y. Y. et al., "Perllm: Personalized inference scheduling with edge-cloud collaboration for diverse llm services," 2024. [Online]. Available: https://arxiv.org/abs/2405.14636

[17] C. Fu, R. Zhang, and Z. W. et al., "A challenger to gpt-4v? early explorations of gemini in visual expertise," 2023. [Online]. Available: https://arxiv.org/abs/2312.12436
[18] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," 2021. [Online]. Available: https://arxiv.org/abs/2103.00020
[19] H. Touvron, L. Martin, and K. S. et al., "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288
[20] X. Chu, L. Qiao, and X. L. et al., "Mobilevlm : A fast, strong and open vision language assistant for mobile devices," 2023. [Online]. Available: https://arxiv.org/abs/2312.16886
[21] NVIDIA, "NVIDIA Jetson Embedded systems," 2024, accessed: September 26, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/
[22] NVIDIA, "NVIDIA Jetson Orin Nano," 2024, accessed: September 26, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
[23] ——, "NVIDIA Jetson Orin AGX," 2024, accessed: September 26, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/
[24] N. B.-G. et al., "Visual riddles: a commonsense and world knowledge challenge for large vision and language models," 2024. [Online]. Available: https://arxiv.org/abs/2407.19474
[25] vik, "moondream2 (revision 92d3d73)," 2024. [Online]. Available: https://huggingface.co/vikhyatk/moondream2
[26] S. Gunasekar and Y. Z. et al., "Textbooks are all you need," 2023. [Online]. Available: https://arxiv.org/abs/2306.11644
[27] yuanz, "llava_qwen15_4b_chat_openai_clip_vit_large_patch14_336 (revision 5070a27)," 2024. [Online]. Available: "https://huggingface.co/yuanzhoulvpi/llava_qwen15-4b-chat_openai-clip-vit-large-patch14-336"
[28] J. Bai, S. Bai, and Y. C. et al., "Qwen technical report," 2023. [Online]. Available: https://arxiv.org/abs/2309.16609