Efficient Vision Knowledge Pipeline Systems on Resource-Constrained Edge Devices

Joshua Scarpinato*, Bobin Deng†, Md Romyull Islam†, Xinyue Zhang†, Kun Suo†
*Southern Polytechnic College of Engineering and Engineering Technology, Kennesaw State University, Georgia, USA
jscarpi1@students.kennesaw.edu
†College of Computing and Software Engineering, Kennesaw State University, Georgia, USA
bdeng2@kennesaw.edu, mislam22@students.kennesaw.edu, {xzhang48, ksuo}@kennesaw.edu

***Abstract*—Empowering local vision and reasoning capability to edge or IoT devices can lower response latency, mitigate processing burden within data centers, cut down network bandwidth cost, and provide stronger privacy safeguards. The success of these edge AI deployments may offer advantages to various applications, such as smart manufacturing, smart agriculture, and autonomous driving. Vision Language Model (VLM) is one of the promising techniques that serves as a foundation module of this development. However, VLM generally requires more hardware resources than a lightweight Vision-CNN plus a Small Language Model (SLM) with similar knowledge reasoning ability. Mid-end edge devices today may not provide sufficient resources to serve most VLMs, especially for the applications that require quick or even real-time responses. In this paper, we develop a flexibility framework for Vision Knowledge Pipeline Systems (VKPS). Unlike VLM, which requires a tightly coupled image encoder, VKPS allows flexibility in replacing the vision functionality with a lightweight Vision-CNN model. Our empirical experiments demonstrate the feasibility and efficiency of VKPS on edge devices compared to VLMs, achieving 2.37X-36.86X speedup in their pre-processing stage across all the testing edge devices. Our analysis indicates that we should increase the hardware RAM size or utilize VKPS focusing on specific vision tasks for efficient vision and reasoning capability on edge devices.**

***Index Terms*—Vision Knowledge Pipeline Systems, Edge Devices, Vision Language Models, Edge AI**

I. Introduction

Implementing AI processing ability on local edge/IoT devices or systems has recently gained more attention from the research community. The benefits of deploying local AI include faster response times, working without network connectivity, and more robust privacy protection. Moreover, local AI processing can lessen the computing workload on the remote servers. ChatGPT experiences out-of-service events [1], which may be due to the high volume of user requests. However, edge IoT devices may spend a high percentage of their time in the idle status. Moving partial AI computation from servers to edge systems can promote a more balanced distribution of computing resources within the entire computing ecosystem.

Several studies have analyzed and implemented Large Language Models (LLM), Small Language Models (SLM), and Vision Language Models (VLM) on the resource-constrained edge systems to obtain benefits [2]–[4]. Adding local vision AI to the system can enhance its processing ability in complex tasks.Unlike LLMs/SLMs, which aim to understand and reason mainly in human languages, the goal of VLMs is to understand and predict the physical world, which is considered as the foundation of physical AI. Even though VLMs have various architectures and different internal components, their main functionality is to perform object/activity analysis from images or video frames, and offer general reasoning processing in one forward pass via a tight-coupling AI model. However, due to the limited edge resources, most open source VLMs are still challenging to deploy directly, despite the existence of various compression techniques (quantization [5], pruning [6], and knowledge distillation [7]). Instead of having a tight-couple VLM, a Vision Knowledge Pipeline System (VKPS) is a loose-coupling approach that has the potential to offer reasoning on the physical world. Rather than having a heavy image decoder, VKPS can flexibly adopt a lightweight Vision-CNN model and connect it with an SLM. SLM is considered an essential foundation of Agentic AI [8] and AI workflow. According to our systematic study, most current VLMs are still generally too heavy to achieve real-time processing on regular off-the-shelf edge devices. The loose-coupling modules of VKPS enable us to flexibly queue the detected frames or combine them to better meet the real-time requirement while maintaining a good reasoning capability.

Integrating smaller Vision-CNN model(s) doesn't indicate the accuracy degradation. For example, by executing the coco-detection dataset, *Florence-2-L* with 0.77B parameters, can only get 43.4 mAP [9], where *YOLO11s* (Vision-CNN) can achieve 47 mAP [10] with only 9.4M parameters. The primary constraint of the VKPS is that the output from the Vision-CNN model may be limited when compared with VLMs, and Vision-CNN is generally challenging in providing systematic information from images or video frames. For example, the YOLO model can detect objects in pre-defined classifications and may ignore other details. Therefore, both VLM and VKPS have their preferred applications. However, considering the challenges of executing VLM locally with fast response, VKPS could be a potential alternative to simultaneously allow edge devices to have vision and knowledge reasoning capability.

In this work, we will systematically study the resource usage of mid-end edge devices when running VLM and VKPS. Aiming to quantify how much speedup the VKPS can achieve compared to VLM. We will provide guidelines and summarization for future edge device manufacturing to ensure more efficient

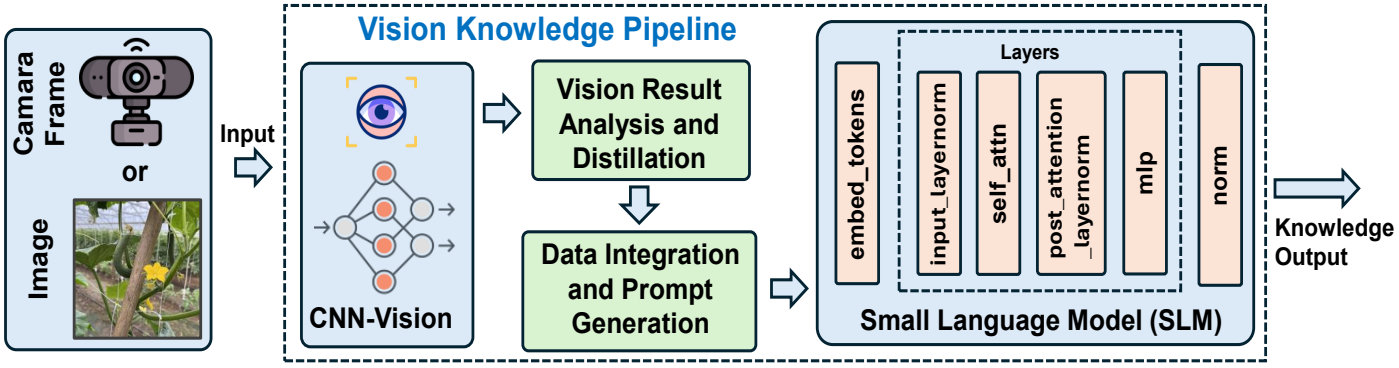

Figure 1: The Framework Overview of Vision Knowledge Pipeline Systems (VKPS)

VLM and VKPS execution. The conclusions from our work will also benefit agentic AI development, especially in model selection strategy. For example, when running a resource-aware AI agent on an edge device, the system should be more likely to build a lightweight VKPS than calling VLM directly. Moreover, our work may contribute to smart manufacturing. Digital twins combined with smart IoT devices can monitor production lines, and offer future event prediction (e.g., high temperature in one unit may lead to robot arm failure in the next 2 hours) or production line simulation. The VLM or VKPS is a foundation component for developing these smart IoT systems.

The key contributions of this paper:

1) We develop a Vision-Knowledge Pipeline Systems (VKPS) framework that can flexibly and easily replace the Vision-CNN and SLM. This framework allows us to offer vision and knowledge reasoning ability on resource-constrained edge devices simultaneously.
2) We systematically evaluate the resource usage of edge devices when executing both VLM and VKPS. We also identify the performance bottlenecks and quantify the speedup of the VKPS approach.
3) Our empirical study provides the design guidelines for future edge device manufacturing to serve VLM and VKPS better. Our study also has great potential to help the developer of edge agentic AI when determining the model selection strategy.

For the remaining sections of this paper, we will discuss the framework of Vision Knowledge Pipeline Systems (VKPS) in Section II. Section III describes our evaluation methodology, and the experimental result analysis is listed in Section IV. Section V is for related work, and we conclude in Section VI.

II. Implementation of Vision Knowledge Pipeline Systems (VKPS)

Vision Knowledge Pipeline System (VKPS) effectively reduces computing costs while maintaining good vision detection and recognition ability. The VKPS can be even further compressed by applying conventional AI model compression techniques, such as quantization, pruning, and knowledge distillation. However, this lightweight benefit from VKPS required AI system developers to put extra effort into manually constructing the vision knowledge pipeline, and partially lost general vision analysis ability. Most edge AI applications typically have clear, specific vision tasks that a lightweight vision-CNN can solve and may potentially skip LVM's heavy image encoder. Therefore, the VKPS is a practical approach to expand physical AI to edge processing.

To provide better flexibility and make VKPS easier to build, we developed a general framework that allows for easy replacement of the vision-CNN or SLM component based on specific requirements. Figure 1 demonstrates the framework overview of general VKPS. The input of the VKPS can be either a static image or sequential frames from a live stream camera (video). Within the vision knowledge pipeline, we will need extra functional components to connect CNN-Vision and SLM. *Vision Result Analysis and Distillation* unit analyzes the output from CNN-Vision and distills the required/useful information. *Data Integration and Prompt Generation* unit uses the distilled information to assemble the prompt and forward it to SLM for knowledge and reasoning results.

III. Evaluation Methodology

This section introduces the methodology to evaluate our Vision Knowledge Pipeline Systems (VKPS) and Vision Language Models (VLMs), including edge hardware platforms, evaluating AI models, benchmarks, and other assistive tools.

A. Edge Hardware Platforms

We select the Raspberry PI 5 and NVIDIA Jetson Orin Nano to represent the off-the-shelf edge devices. Raspberry PI 5 contains 4 CPU cores with a 2.4 GHz frequency each. Like other low-end or mid-end edge devices, the default Raspberry PI 5 is not integrated with a GPU. According to Dhar et al. [2], memory is the bottleneck when executing the *LLaMa-2* 7B with INT4 Quantization. Therefore, we evaluate multiple versions of Raspberry PI 5 with different RAM sizes. NVIDIA Jetson Orin Nano consists of 6 CPU cores and 1024 GPU cores. Figure 2 is a screenshot of our hardware platforms for evaluations, and partial parameters are summarized in Table I.

B. AI Models

Table II lists the AI models we evaluate. The popular YOLO (You Only Look Once; YOLOv8n) [11], which is mainly used

Table I: Overview of Edge Hardware Platforms

Name	RAM	CPU	GPU
Raspberry PI 5	2 GB	2.4GHz, 4 cores	N/A
Raspberry PI 5	4 GB	2.4GHz, 4 cores	N/A
Raspberry PI 5	8 GB	2.4GHz, 4 cores	N/A
Raspberry PI 5	16 GB	2.4GHz, 4 cores	N/A
Jetson Orin Nano	8 GB	1.7GHz, 6 cores	1020MHz, 1024 cores

Table II: Vision-CNNs, SLMs, and VLMs For Evaluations

Model Name	Number of Parameters	Size	Category
Yolov8n	2.6 M	6.25 MiB	Vision-CNN
Gemma3	1 B	1.87 GiB (f16)	SLM
SmolVLM2	2.2 B	2.34 GiB (Q8_0)	VLM
Llama3.2	1 B	1.2 GiB (Q8_0)	SLM
nanoLLaVA-1.5	1 B	2.0 GiB (f16)	VLM

Table III: Latency Comparison between a VLM (SmolVLM2) and a VKPS (YOLO+Gemma3) in *VanaHighlights*

Config	SmolVLM2 (millisecond)					YOLO+Gemma3 (millisecond)			Speedup
	Img Encode	Img Decode	Load	Prompt Eval	Time Sum	Load	Prompt Eval	Time Sum	
PI_2GB	10024.00	38464.00	41851.29	99605.07	189944.36	46677.90	16592.24	63270.14	3.00
PI_4GB	12316.58	352.37	1528.79	18003.26	35351.00	1142.36	479.92	1622.29	21.79
PI_8GB	12033.00	3336.47	690.58	16828.06	32888.11	865.30	434.35	1299.65	25.31
PI_16GB	14294.05	3976.32	732.30	19919.17	38921.85	859.53	491.13	1350.66	28.82
Jetson_Orin_Nano	1016.79	9.95	15909.97	14083.18	31019.90	5570.37	1822.40	7392.76	4.20

Table IV: Latency Comparison between a VLM (SmolVLM2) and a VKPS (YOLO+Gemma3) in *public-test-images*

Config	SmolVLM2 (millisecond)					YOLO+Gemma3 (millisecond)			Speedup
	Img Encode	Img Decode	Load	Prompt Eval	Time Sum	Load	Prompt Eval	Time Sum	
PI_2GB	44554.00	20046.00	53915.88	177282.66	295798.54	33965.76	15520.65	49486.41	5.98
PI_4GB	13311.33	3705.29	1452.65	19802.57	38271.83	914.87	497.40	1412.27	27.10
PI_8GB	12656.85	3544.95	715.89	17761.43	34679.12	866.58	452.67	1319.25	26.29
PI_16GB	14842.76	4160.75	761.73	20727.86	40493.10	886.50	494.33	1380.83	29.33
Jetson_Orin_Nano	882.60	10.22	11033.48	6115.98	18042.28	5547.10	2081.31	7628.42	2.37

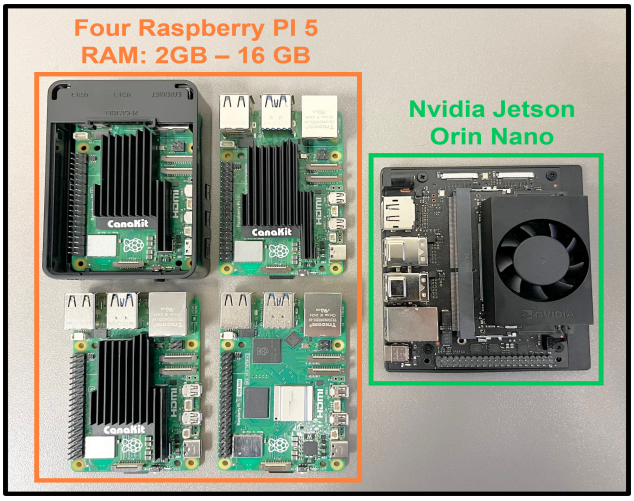

Figure 2: Edge Hardware Platforms

for object detection and segmentation, is selected to present the vision-CNN model in our evaluations. The SLMs are represented by the widely used models Gemma-3 (gemma-3-1b) and Llama-3.2 (llama-3.2-1b) for knowledge checking and reasoning. YOLOv8 is connected with Gemma-3 or Llama-3.2 to construct the VKPS that were introduced in Section II. The VLMs are represented by SmolVLM2 or nanoLLaVA-1.5, which is designed for IoT device deployment. VLMs generally consist of two parts, an image encoder and a language decoder. Therefore, the size of VLM should be the sum of both parts. The size of SmolVLM2 is *2.34* GiB, where its image encoder is *565* MiB and language decoder is *1.79* GiB. Similarly, the size of nanoLLaVA-1.5 is *2.0* GiB, where its image encoder is *822* MiB and language decoder is *1.2* GiB. To ensure fair comparison between VKPS and VLMs, the sizes of knowledge reasoning components after applying compression techniques (e.g, quantization) should match as closely as possible. For example, gemma-3-1b (f16) of VKPS is *1.87* GiB and the language decoder of SmolVLM2 is *1.79* GiB. Similarly, llama-3.2-1b (Q8_0) of VKPS is *1.2* GiB and the language decoder of nanoLLaVA-1.5 is also *1.2* GiB.

C. Benchmarks and Other Tools

We select two life domian datasets, *Egoshots* [12] and *KTVIC* [13], as benhmark for evaluations. To lower the evaluation time on resource-constrained edge devices, in this work, we test VanaHighlights from *Egoshots* and the first 10% of *public-test-images* from *KTVIC*. Due to similarity of activity pattern, for Raspberry PI 5 with 2GB RAM, we reduce the test set to 10% and 0.3% for *VanaHighlights* and *public-test-images*, respectively. We also utilize the nmon tool to monitor the resource usage and collect corresponding data for analysis.

IV. Experimental Results and Summarizations

This section systematically evaluates the resource utilization based on response latency, CPU utilization, and memory utilization. According to our empirical analysis, we will provide guidelines and a summary that may help the IT community develop efficient vision knowledge AI systems on resource-constrained edge devices.

Table V: Latency Comparison between a VLM (nanoLLaVA) and a VKPS (YOLO+Llama3.2) in *VanaHighlights*

Config	nanoLLaVA (millisecond)					YOLO+Llama3.2 (millisecond)			Speedup
	Img Encode	Img Decode	Load	Prompt Eval	Time Sum	Load	Prompt Eval	Time Sum	
PI_2GB	9210.00	18813.00	15995.63	45829.20	89847.83	8670.69	365.97	9036.66	9.94
PI_4GB	11709.53	7691.37	973.36	19761.22	40135.47	719.54	506.36	1225.90	32.74
PI_8GB	12105.16	7878.37	725.64	20354.27	41063.43	701.87	482.45	1184.32	34.67
PI_16GB	12823.79	8450.47	764.60	21658.45	43697.32	708.89	500.22	1209.11	36.14
Jetson_Orin_Nano	738.11	394.42	3011.07	3139.40	7282.99	1647.58	1314.79	2962.37	2.46

Table VI: Latency Comparison between a VLM (nanoLLaVA) and a VKPS (YOLO+Llama3.2) in *public-test-images*

Config	nanoLLaVA (millisecond)					YOLO+Llama3.2 (millisecond)			Speedup
	Img Encode	Img Decode	Load	Prompt Eval	Time Sum	Load	Prompt Eval	Time Sum	
PI_2GB	10340.00	19762.00	35887.09	49792.48	115781.57	13194.23	1075.34	14269.57	8.11
PI_4GB	12649.55	8277.98	872.32	21313.77	43113.61	718.40	535.93	1254.33	34.37
PI_8GB	12137.49	7920.13	726.33	20433.80	41217.75	711.76	490.71	1202.47	34.28
PI_16GB	14067.73	9345.85	858.48	23842.77	48114.83	731.04	574.39	1305.43	36.86
Jetson_Orin_Nano	729.62	393.09	3469.53	3082.83	7675.07	1740.67	1307.95	3048.63	2.52

Table VII: Average *Token Per Second* Comparison for Output Content Generation (*VanaHighlights*)

Config	Ave Tokens per Second (SmolVLM2)	Ave Tokens per Second (YOLO+Gemma3)	Ave Tokens per Second (nanoLLaVA)	Ave Tokens per Second (YOLO+Llama3.2)
PI_2 GB	0.040000	0.050000	0.070000	7.240000
PI_4 GB	4.671579	4.560000	7.985263	7.322632
PI_8 GB	4.779474	4.434211	8.042105	7.764211
PI_16 GB	4.641579	4.676842	8.030000	7.308421
Jetson_Nano	12.986842	12.051579	16.636842	23.802105

Table VIII: Average *Token Per Second* Comparison for Output Content Generation (*public-test-images*)

Config	Ave Tokens per Second (SmolVLM2)	Ave Tokens per Second (YOLO+Gemma3)	Ave Tokens per Second (nanoLLaVA)	Ave Tokens per Second (YOLO+Llama3.2)
PI_2 GB	0.030000	0.050000	0.020000	7.590000
PI_4 GB	4.406182	4.421636	7.760364	7.006000
PI_8 GB	4.471636	4.626545	8.036000	7.283455
PI_16 GB	4.367455	4.524727	7.728182	6.923273
Jetson_Nano	13.585455	11.870545	16.622182	23.795636

A. Response Latency

Observation: Table III, Table IV, Table V, Table VI, Table VII and Table VIII demonstrate the latency comparison between a VLM and a Vision Knowledge Pipeline System (VKPS) when executing the *VanaHighlights* from *Egoshots* and partial *public-test-images* from *KTVIC*. The evaluated VLM is either *SmolVLM2* or *nanoLLaVA-1.5*. Similarly, the evaluated VKPS is either *YOLOv8n+Gemma3* or *YOLOv8n+Llama3.2*. Different generative AI models usually generate non-identical content for the same input prompt and require different numbers of tokens. To make a fair comparison, we divide the latency evaluation into two stages: (1) the *pre-processing stage* and (2) the *content generation stage*. The *pre-processing stage* includes *image encode*, *image decode*, *load*, and *prompt evaluation* for VLM and contains only *load* and *prompt evaluation* for VKPS. The *Time Sum* columns in Table III, Table IV , Table V and Table VI are the total average latency of the *pre-processing stage*. The *Speedup* column shows the speed that the VKPS can achieve when compared with VLM in the *pre-processing stage*. We observe the significant speedup range from *2.37X* to *36.86X*. Table VII and Table VIII compare the *average token per second* between VLM and VKPS (two sets) on two benchmarks (VanaHighlights and public-test-images). Because the sizes of the generative models we selected are similar, the token per second results we observe are close. One interesting comparison of *Average Tokens per Second* between *nanoLLaVA-1.5* and *YOLOv8n+Llama3.2* is in the PI_2GB configuration. The VKPS is significantly better than VLM because VKPS (*YOLOv8n+Llama3.2*) can be totally stored within RAM while VLM (*nanoLLaVA-1.5*) cannot. Therefore, utilizing the VKPS can significantly lower response time unless the model generates long outputs.

Sub-Section Summarization 1: Compared with VLM, VKPS have great potential to lower response time on edge systems and more effectively satisfy real-time requirements.

B. CPU Utilization

Observation: Figure 3 and Figure 4 show the CPU utilization comparison between VLM and VLPS across testing edge devices. We use (*SmolVLM2* and *nanoLLaVA-1.5*) to represent regular edge VLM, and (*YOLO+Gemma3* and *YOLO+Llama3.2*) to represent the VLPS in the remaining parts of this section. Considering the similarity in activity patterns, we capture the first 1000 seconds for all the subfigures.

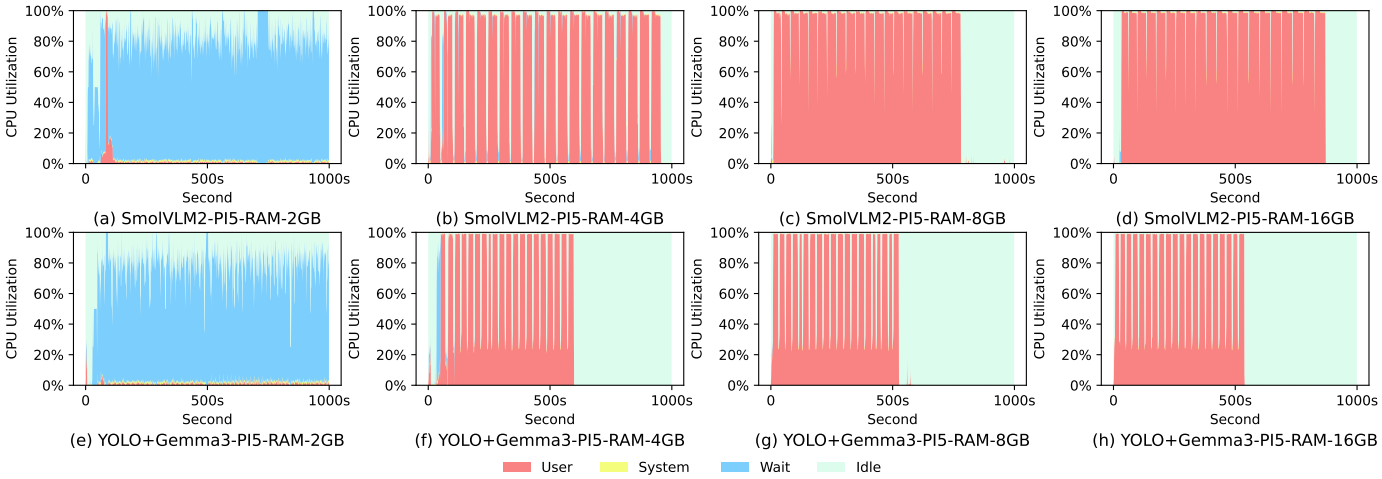

Figure 3: CPU Utilization Comparison Between *SmolVLM2* (VLM) and *YOLOv8+Gemma3* (VKPS)

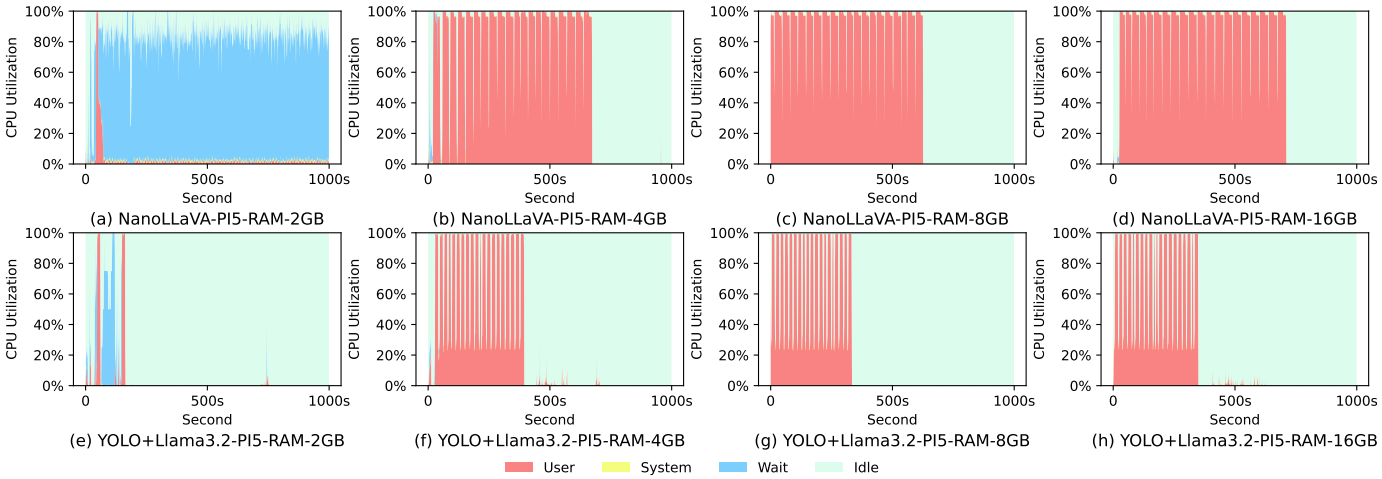

Figure 4: CPU Utilization Comparison Between *nanoLLaVA* (VLM) and *YOLOv8+Llama3.2* (VKPS)

For the Raspberry PI with 2 GB RAM, we are almost unable to see the user utilization where our AI application overhead belongs, while waiting time dominates the utilization percentage in *nanoLLaVA-1.5*, *SmolVLM2*, and *YOLO+Gemma3*. The main reason is that the hardware platform with 2GB RAM is insufficient to maintain *nanoLLaVA-1.5*, *SmolVLM2*, or *YOLO+Gemma3* residing within memory. However, 2GB RAM is sufficient for *YOLO+Llama3.2* (more details are available in Section IV-C.). If we don't have sufficient RAM, the system often kicks out and fetches back data from the disk or SSD, significantly slowing down the system's response time. Because of this long waiting time, the processing unit of Raspberry PI has to wait for the data to arrive before it can move forward. For the hardware platforms with RAM sizes of 4GB, 8GB, and 16GB, we observe that the CPU utilization periodically reaches 100%. After completing one image/frame and before starting to process a new image/frame, the CPU utilization will temporarily drop. If we compare *SmolVLM2* and *YOLO+Gemma3* behaviors in the same RAM size, we observe that the execution times of *YOLO+Gemma3* are always less than *SmolVLM2*. We observe similar patterns when comparing *nanoLLaVA-1.5* and *YOLO+Llana3.2*. Therefore, CPU utilization results also indicate that VKPS requires a lower response time than VLM.

Sub-Section Summarization 2: If the size of hardware RAM is less than the vision AI model, the CPU may always be waiting for data from the slow disk/SSD, and therefore, the RAM/memory becomes the performance bottleneck. We can provide sufficient hardware memory or compress the AI model for edge deployment to reduce the disk/SSD access frequency.

C. Memory Utilization

Observation: Memory is identified as the bottleneck for regular LLM or SLM edge deployment [2], [3]. Considering the similarity in AI model architectures, we analyze the memory footprints of LVM and VKPS in this section. The memory activity comparison of VLM (*SmolVLM2* and *nanoLLaVA-1.5*) and VKPS (*YOLO+Gemma3* and *YOLO+Llana3.2*) are shown in Firgure 5 and Firgure 6. In the Raspberry PI with 2 GB RAM (e.g., Figure 5 (a) and Figure 5 (e)), the free memory (green line; memfree) is consistently close to 0 in both VLM and VKPS. This observation provides clearer evidence of high response latency (Section IV-A) and high waiting time

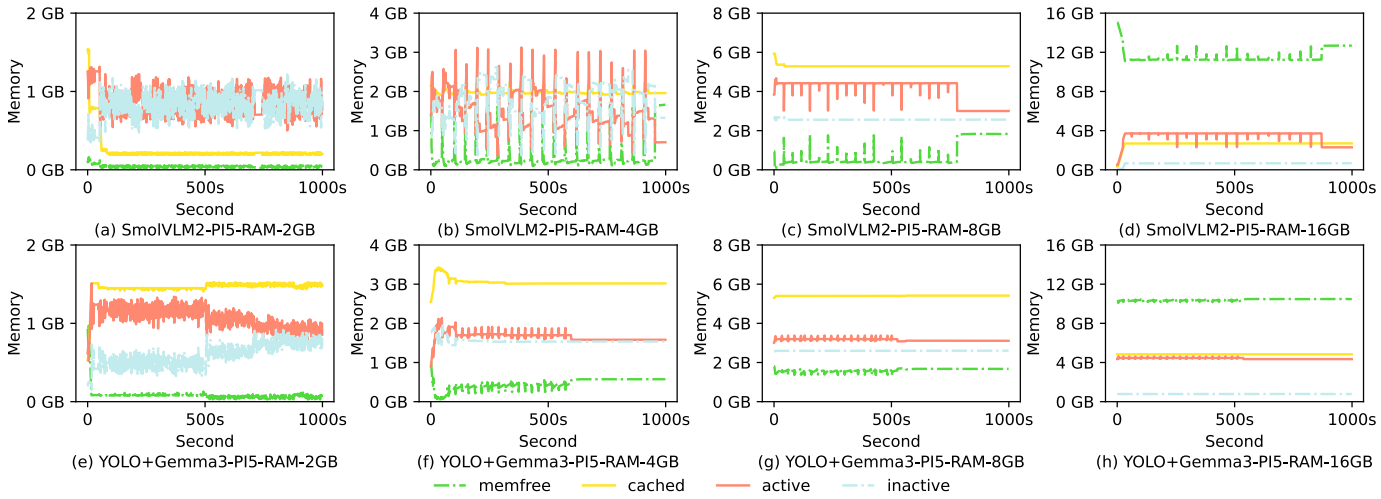

Figure 5: Memory Utilization Comparison Between *SmolVLM2* (VLM) and *YOLOv8+Gemma3* (VKPS)

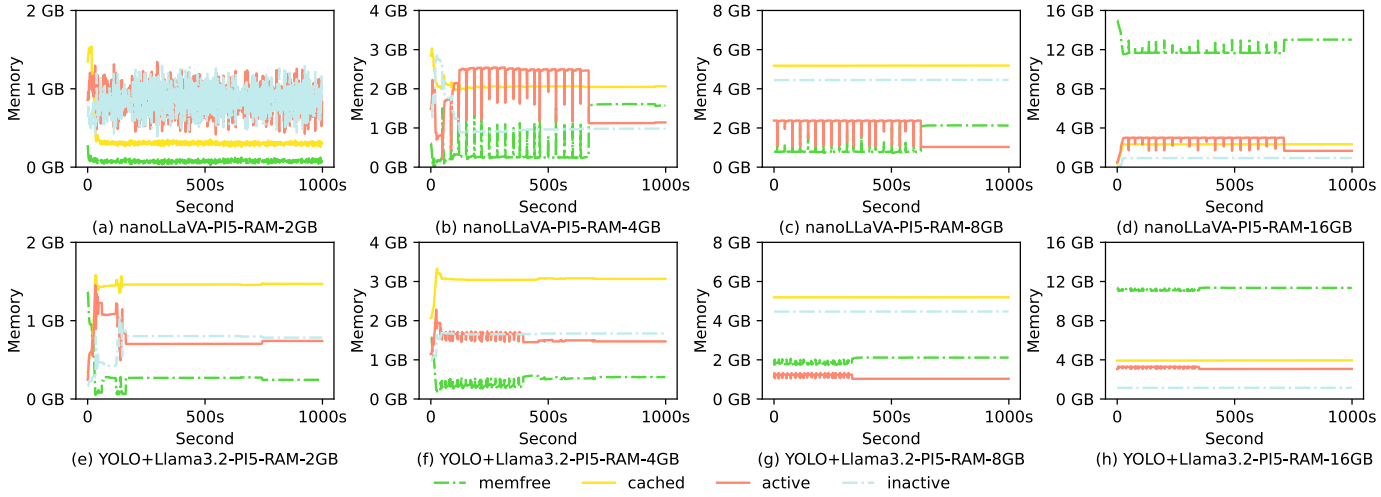

Figure 6: Memory Utilization Comparison Between *nanoLLaVA* (VLM) and *YOLOv8+Llama3.2* (VKPS)

percentage (Section IV-B) in Raspberry PI with 2GB RAM. However, in Figure 6 (e), the free memory is not very close to 0 in most of the time and this can explain why the Ave Tokens per Second of *YOLO+Llana3.2* is much better than that of *nanoLLaVA* in the PI_2GB rows of Table VII and Table VIII. In the hardware platform with 4GB and 8GB RAM, VKPS (*YOLO+Gemma3*) has more free memory than VLM (*SmolVLM2*). The main reason is that, unlike regular VLMs, VKPS generally does not require a heavy image/frame encoder. In the Raspberry PI with 16GB RAM, both VKPS and VLM have high percentages of free memory.

Sub-Section Summarization 3: Insufficient memory is one of the main obstacles that slow down the edge vision AI. Increasing the RAM size is an effective way to provide knowledge and reasoning capability to an edge system. If the RAM size expansion is too costly or infeasible, building a VKPS with lightweight Vision-CNN could be an alternative for faster processing while maintaining the specific vision functionality.

D. Summarization

Deploying VLM on off-the-shelf edge devices may still be challenging in meeting real-time requirements. For edge devices with less than 2GB RAM, achieving acceptable response time and quality of service is difficult. Compared with VLM, VKPS may be a better way to achieve edge devices' vision and knowledge reasoning capability efficiently, possibly sacrificing some flexibility and details from input images or videos.

A large RAM size for the edge device hardware is crucial to ensure a lower response time if we want to support both vision and knowledge reasoning effectively. If the hardware platform is unable to reconfigure or has budget considerations, VKPS may be a better option for an edge device or system to integrate vision and knowledge reasoning capability. For the agentic AI deployment on edge, we should prioritize selecting Vision-CNN+SLM instead of heavier VLM.

V. Related Work

Vision encoders are the key element behind VLMs. The idea of vision transformer(ViT), a vision encoder, changed the path of multimodal AI [14]. Instead of convolution, it divides images into patches and treats them like tokens in a transformer model. As a result, large pretrained vision models such as CLIP emerged [15]. Recently, models like Llava have become popular, connecting a vision encoder and an LLM for general-purpose visual and language understanding into an LLM [16]. However, these VLMs still exhibit high latency and memory demands, particularly when GPUs are unavailable [2]. Our work differs by avoiding heavy vision encoders, and combining a lightweight Vision-CNN as an alternative.

Decoupled pipelines separate the vision and reasoning components into modular units. This typically combines a lightweight vision model for object detection and an SLM for reasoning and response generation. Unlike PLACE [17] and BLIP [18], which tightly couple vision and language in a single pre-trained model for image synthesis and vision-language understanding, respectively, our approach decouples the vision and reasoning stages into modular components (e.g., YOLO [11]+ SLM [8]), enabling more flexibility and efficiency for edge deployment. Most high-performing VLMs or multi-modal models are huge (e.g., GPT-4) and cannot run without cloud infrastructure [19]. The available lightweight VLMs face challenges on constrained edge devices such as RaspberryPi related to limited memory, computational capacity, and energy efficiency [4], [20]. However, our proposed framework combines a very lightweight and precise Vision-CNN model and an SLM that can generate faster and accurate results.

VI. Conclusion

Pushing vision and knowledge reasoning AI from the server to the edge or IoT devices can gain considerable rewards, including potentially less response latency, more robust privacy protection, working without network connectivity, and less processing burden on the remote servers. However, due to limited resources, most VLMs are challenging to execute effectively on mid-end off-the-shelf edge devices. VKPS could be an alternative to extend this AI capability to edge or IoT systems for applications in the physical world. We develop a flexible and effective VKPS framework to facilitate the AI deployment while adapting to different constraints. Our empirical experiments and analysis show that the VKPS can significantly reduce the response time, although it may partly sacrifice general detection ability. VKPS offers better opportunities for resource-constrained edge devices to simultaneously have vision and knowledge reasoning abilities. Our findings provide guidelines for future edge device design, agentic AI development, and smart manufacturing.

Acknowledgement

We appreciate that anonymous reviewers reviewed our paper. This research is partially supported by the following US National Science Foundation grants: NSF-2348417, NSF-2431597, and NSF-2210744.

References

[1] C. Eudaily. (2025) Chatgpt outage: Openai reports 'elevated error rates' for several hours. [Online]. Available: https://www.cnbc.com/2025/06/10/chatgpt-down-outage.html

[2] N. Dhar, B. Deng, D. Lo, X. Wu, L. Zhao, and K. Suo, "An empirical analysis and resource footprint study of deploying large language models on edge devices," in *Proceedings of the 2024 ACM southeast conference*, 2024, pp. 69–76.

[3] M. R. Islam, N. Dhar, B. Deng, T. N. Nguyen, S. He, and K. Suo, "Characterizing and understanding the performance of small language models on edge devices," in *2024 IEEE International Performance, Computing, and Communications Conference (IPCCC)*. IEEE, 2024, pp. 1–10.

[4] A. Sharshar, L. U. Khan, W. Ullah, and M. Guizani, "Vision-language models for edge networks: A comprehensive survey," *IEEE Internet of Things Journal*, 2025.

[5] J. Lang, Z. Guo, and S. Huang, "A comprehensive study on quantization techniques for large language models," in *2024 4th International Conference on Artificial Intelligence, Robotics, and Communication (ICAIRC)*. IEEE, 2024, pp. 224–231.

[6] X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.

[7] X. Xu, M. Li, C. Tao, T. Shen, R. Cheng, J. Li, C. Xu, D. Tao, and T. Zhou, "A survey on knowledge distillation of large language models," *arXiv preprint arXiv:2402.13116*, 2024.

[8] P. Belcak, G. Heinrich, S. Diao, Y. Fu, X. Dong, S. Muralidharan, Y. C. Lin, and P. Molchanov, "Small language models are the future of agentic ai," *arXiv preprint arXiv:2506.02153*, 2025.

[9] B. Xiao, H. Wu, W. Xu, X. Dai, H. Hu, Y. Lu, M. Zeng, C. Liu, and L. Yuan, "Florence-2: Advancing a unified representation for a variety of vision tasks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 4818–4829.

[10] (2025) Ultralytics yolo11. [Online]. Available: https://docs.ultralytics.com/models/yolo11/

[11] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[12] P. Agarwal, A. Betancourt, V. Panagiotou, and N. Díaz-Rodríguez, "Egoshots, an ego-vision life-logging dataset and semantic fidelity metric to evaluate diversity in image captioning models," in *Machine Learning in Real Life (ML-IRL) Workshop at the International Conference on Learning Representations (ICLR)*, Mar 2020. [Online]. Available: https://arxiv.org/abs/2003.11743

[13] A.-C. Pham, V.-Q. Nguyen, T.-H. Vuong, and Q.-T. Ha, "Ktvic: A vietnamese image captioning dataset on the life domain," *arXiv preprint arXiv:2401.08100*, 2024.

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

[15] 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.

[16] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, pp. 34 892–34 916, 2023.

[17] Z. Lv, Y. Wei, W. Zuo, and K.-Y. K. Wong, "Place: Adaptive layout-semantic fusion for semantic image synthesis," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 9264–9274.

[18] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International conference on machine learning*. PMLR, 2022, pp. 12 888–12 900.

[19] R. OpenAI, "Gpt-4 technical report. arxiv 2303.08774," *View in Article*, vol. 2, no. 5, p. 1, 2023.

[20] S. Lou, S. Ge, J. Yu, and G. Zhang, "Tinyvision: Distributed vision-language model with efficiency and privacy for edge deployment," in *International Conference on Intelligent Computing*. Springer, 2025, pp. 175–187.