Beyond the Shadows: A Deep Dive into Profiling Modern Mixed-Modal and Multi-modal Transformer Models

Oruç Cakir[1], Julian Pavon[2,3][0000−0002−8291−509X], Betül Aydeğer[1][0009−0003−9117−4750], Zeynep Yavuz[1][0009−0008−3289−675X], Ivan Vargas Valdivieso[0000−0002−5092−3829], Oğuz Ergin[1][0000−0003−2701−3787], Adrian Cristal[2][0000−0003−1277−9296], and Osman Ünsal[2][0000−0002−0544−9697]

[1] TOBB University of Economics and Technology, Ankara, Turkey
{initial,surname}@etu.edu.tr
[2] Barcelona Supercomputing Center, Barcelona, España
{firstname.surname}@bsc.es
[3] Universitat Polytecnica de Catalunya, Barcelona, España
{firstname.surname}@upc.edu

Abstract. The rapid advancement of Large Language Models (LLMs) has led to increasingly sophisticated architectures, transitioning from text-based models to multi-modal and mixed-modal systems. These advancements raise critical questions about model accuracy, execution efficiency, and hardware performance. While Python-based frameworks such as PyTorch and Hugging Face provide high-level benchmarking tools, they lack the hardware insights needed for optimizing performance. Conversely, hardware-level profilers like perf and NVIDIA nsys offer hardware performance metrics but are less adaptable to emerging models.
To bridge this gap, we introduce Lumina, a unified framework that combines Python's flexibility with C++'s hardware control to provide a comprehensive evaluation of LLMs. Lumina integrates profiling tools like perf, PAPI, nsys, and ncu to assess both functional accuracy and computational efficiency across CPU and GPU platforms. We demonstrate the effectiveness of Lumina by profiling LLaVA-Mistral-7B, Chameleon-7B, and Deepseek-Janus-Pro-7B, showing that on average, Deepseek-Janus-Pro-7B executes 5.7× faster the others models. Additionally, we present the first in-depth hardware profiling analysis of mixed-modal models, offering insights into their computational demands and optimization opportunities. Lumina ensures compatibility with evolving LLM architectures, making it a versatile tool for researchers and practitioners optimizing next-generation AI models.

Keywords: Artificial Intelligence · Mixed-modal · Multi-modal · Benchmarking · Performance Evaluation · GPU Profiling · CPU Profiling

1 Introduction

Recent years have seen explosive growth in Large Language Models (LLMs), with increasing model sizes, higher accuracy, and diverse architectures. LLMs

have evolved from pure natural language processing (NLP) [14], to optical character recognition (OCR) [6], and generative AI [14]. Starting with single-expert architectures [14,19] designed for specific tasks, followed by multi-modal systems that integrate multiple experts to handle combined text and image inputs. These systems use techniques like cross-modal fusion [9] to merge data across experts.

More recently, mixed-modal architectures [2, 16] have raised the bar by enabling unified processing of both text and image inputs. This early fusion improves LLMs performance and cross-modal understanding, and eliminates output misalignment seen in multi-modal models. By processing both modalities simultaneously, mixed-modal systems offer better reasoning capabilities.

With the ever-expanding landscape of LLMs —illustrated by the variety of GPT and LLaMA models —it is increasingly important to assess the performance of such models in today's computing systems. New models like Chameleon and Deepseek, and the shift from multi-modal to mixed-modal architectures, prompt questions regarding their functionality and performance: (i) accuracy, (ii) execution time, (iii) size, (iv) performance bottlenecks, among others. Answering these questions is critical for selecting the appropriate model for deployment.

Python-based tools [22] dominate LLM evaluation due to their robust ecosystem, including libraries like PyTorch [17], TensorFlow [1] and Hugging Face Transformers [22], which simplify AI model development. These tools evaluate high-level performance metrics such as latency, memory consumption, and computation cost. However, they often fall short in providing hardware-level insights, overlooking performance bottlenecks like the overhead of specific CPU instructions or GPU thread utilization, which are vital for optimizing LLM execution.

At the hardware level, analyzing parameters such as memory bandwidth, cache utilization, and branch prediction behavior becomes essential to optimize LLM performance. Efficient memory access can significantly reduce cache misses, which in turn can decrease latency and reduce energy consumption. On specialized hardware such as GPUs or AI accelerators, efficient memory management plays a key role in maximizing performance. On edge devices with limited power and memory resources, optimizing CPU-specific features such as cache utilization and quantized execution (e.g., INT8/INT4) is crucial to improve the efficiency of LLM inference. Additionally, as open-source ISAs like RISC-V gain traction, profiling CPU performance becomes more important for identifying bottlenecks and optimizing instruction sets tailored to the specific needs of LLM workloads.

Hardware profilers, such as perf [12] and NVidia Nsights [12], are used to asses the low hardware-level detail in LLM performance analysis. However, these profilers often struggle with evaluating large-scale, complex workloads like LLMs. This is because the high abstraction level of LLMs, which rely on intricate layers and large parameter spaces, makes it difficult to map the behavior of each component directly to low-level hardware details.

While Python-based tools are highly flexible and allow easy integration of new models and features, they fail to provide the detailed hardware-level analysis required for an in-depth performance evaluation of LLMs. On the other hand, nP-based tools provide essential low-level insights into hardware perfor-

mance, but they lack the flexibility of Python-based frameworks and are less adaptable to the rapid development of new LLM architectures. To address these challenges, we propose **Lumina**, a unified framework for LLM performance evaluation and hardware profiling. Lumina combines Python's flexibility with the low-level hardware and memory management control of C++, enabling detailed insights into both LLM functionality and hardware-level performance. By leveraging tools like perf [13], PAPI [7], nsys [12], and ncu [12], Lumina provides a comprehensive set of hardware performance metrics, helping researchers identify and optimize performance bottlenecks. Lumina is designed to seamlessly adapt to evolving LLM architectures —from traditional text-based models to multi-modal and mixed-modal systems —ensuring compatibility across a wide range of LLMs and hardware systems. With its modular approach, Lumina can be extended to support emerging model formats and architectures, making it a versatile tool for evaluating both functional accuracy and computational efficiency in next-generation AI models.

We make the following key contributions in this work:

- We present Lumina, an unified framework for performance and hardware profiling analysis. Lumina leverages the flexibility of Python and performance orientation of C++ to build a framework capable of (i) working with multiple LLM variants and architectures and, (ii) evaluate multiple low hardware level performance metrics.
- We evaluate the functionality of Lumina by extensively profiling LLaVA-Mistral-7B [10], Meta Chameleon-7B [16] and Deepseek-Janus-Pro-7B [2], one multi-modal and two mixed-modal LLMs, respectively. Our evaluation results demonstrate that Deepseek-Janus-Pro-7B significantly outperform to LLaVA-Mistral-7B and Chameleon-7B by 4.5× and 6.8×, respectively. Moreover, in this analysis we evaluate both CPU and GPU performance, showcasing the flexibility of Lumina not to only work with different LLMs architectures but also different hardware platforms.
- We carried out the first performance and hardware profiling analysis of Chameleon-7B and Deepseek-Janus-Pro-7B, two mixed-modal LLMs recently released by Meta and Deepseek, respectively. Moreover, we present the first OCR-based analysis of the Chameleon-7B and Deepseek-Janus-Pro-7B as well.

2 Related work

2.1 The Recent Development of LLMs

The field of Large Language Models (LLMs) has progressed beyond *uni-modal* architectures, which process only textual inputs, toward *multi-modal* and *mixed-modal* models capable of handling diverse data types. Fig. 1 compares *uni-modal*, *multi-modal* and *mixed-modal* architectures. Early LLMs, such as GPT [14], focused on natural language processing tasks but were inherently limited to textual data [19].

To address these limitations, **multi-modal** models introduced separate vision and text encoders, combining their outputs through late fusion techniques. Models such as CLIP [15] and BLIP-2 [9] enabled text-image reasoning, achieving strong results in tasks like image captioning and visual question answering. However, their modular design restricts fine-grained cross-modal interactions, making it difficult to perform spatial reasoning or interpret embedded text within images [10].

In contrast, **mixed-modal** architectures integrate image and text inputs into a shared token space, allowing continuous interaction throughout the model. Approaches such as Chameleon [16] and Janus-Pro [2] process both modalities within a single transformer, enabling deeper cross-modal dependencies. While this improves performance in vision-language tasks, it increases computational costs and requires large, well-aligned datasets, complicating training and deployment [10, 16].

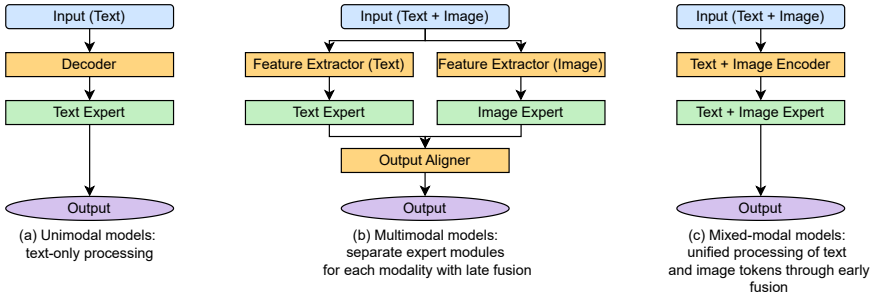

Fig. 1. The evolution of LLM architectures.

2.2 Prior Work on LLM Performance Analysis

Python-based tools. [17] are directly integrated in LLM pipelines alongside AI frameworks such as PyTorch and TensorFlow. These tools leverage built-in profilers (e.g., PyTorch Profiler [17]) and general-purpose performance analysis utilities (e.g., Py-Spy [5]) to monitor GPU utilization, FLOPs, and execution traces. While effective for estimating computational costs, they primarily focus on high-level operations and often lack detailed insights into hardware inefficiencies, such as cache misses, memory stalls, or branch mispredictions.

Hardware profilers. [21] offer fine-grained performance monitoring by interfacing directly with CPU and GPU hardware counters. Tools such as *perf*, PAPI, Intel VTune, and NVIDIA Nsight provide hardware-level metrics on execution pipelines, memory access patterns, and processor utilization. However, the high abstraction level of LLMs makes it difficult to correlate specific model components with these raw hardware metrics, often resulting in incomplete or misinterpreted performance insights.

Despite these efforts, current profiling approaches lack an integrated solution that bridges structured LLM benchmarking with detailed hardware-level

analysis. Our work addresses this gap by introducing a tool that combines high-level model evaluation with in-depth profiling, enabling a more comprehensive understanding of LLM performance across different hardware platforms.

3 Lumina: Architecture and components

Lumina is an unified evaluation and profiling framework carefully designed to bridge the gap between high-level functional metrics and hardware-level performance analysis. It combines Python's high-level functionality with C++'s performance profiling power to deliver detailed insights into both functional and hardware performance of LLMs. As previously commented, hardware profilers lack direct integration with AI frameworks. To address this, Lumina automatizes and facilitate the hardware-level evaluation process, including the setup, benchmarking, results gathering and plotting, and comparison between different LLMs, significantly reducing the manual effort required. Fig. 2 depicts the architecture overview and execution flow of Lumina. Next, we delve into the overall operation flow and components and interconnection of each stage.

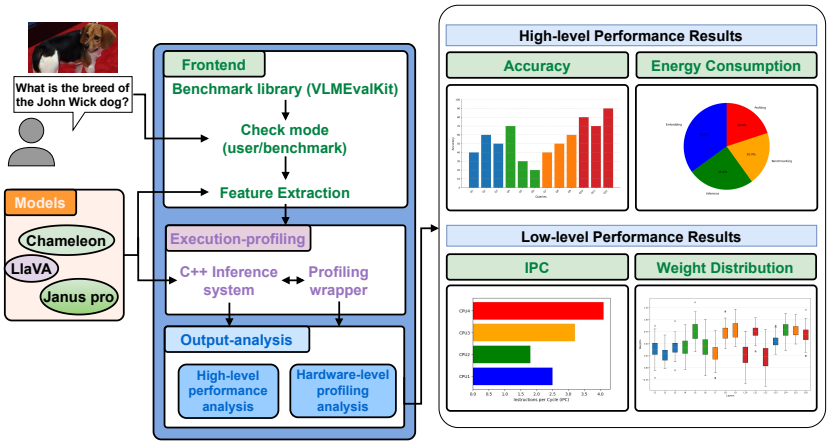

Fig. 2. Lumina architecture overview and execution flow.

3.1 Lumina operation flow

The execution flow in Lumina comprises three key stages:

– **Frontend.** This stage is the interface between the user and the rest of components in Lumina. It works in two modes: (i) user-mode where Lumina processes end-user queries, and (ii) benchmark mode where it executes different benchmarks using a VLMEvalKit library. By doing so, our tool can be easily deploy into servers to evaluate not only benchmarks but also queries from real-users. For every input query the *frontend* generates embeddings using the Transformers library [4] in python.

- **Execution-profiling.** The embeddings generated by the *frontend* are forwarded to this stage. For every set of input embeddings, (i) it executes the inference using a high performance C++-based implementation and (ii) evaluates high-level performance or hardware-level profiling metrics. As deeply explained in §3.3, we have extend the C++-based inference system to support multiple profiling tools enabling Lumina to evaluate a vast number of hardware metrics.
- **Output-analysis.** This stage process the performance and profiling information from the *Execution-profiling* stage and generates read-friendly outputs such as plots and tables.

3.2 Frontend components

The *frontend* is composed of two main modules: the *VLMEvalKit* [3] and a *feature extraction* module.

A. Benchmarking with VLMEvalKit. VLMEvalKit provides standardized benchmarks for evaluating multi-modal models across text, image, and mixed-modal tasks. We extend it with a custom `save-embeddings()` function for each integrated model, enabling extraction of input embeddings required for hardware-level profiling. Additionally, `Lumina` supports quantization-aware benchmarking (4-bit, 8-bit), with automated control of embedding extraction, activation logging, and weight distribution dumping through hook-based flags.

B. Feature extraction. This module transforms raw inputs from `VLMEvalKit` into embeddings using the `transformers` library. By decoupling preprocessing from inference, it allows seamless integration with C++ backends like `LlaMA.Cpp`, which lack native multi-modal preprocessing. This design ensures flexibility and enables efficient profiling for both text-only and mixed-modal models.

3.3 Execution-profiling components

This stage is composed of a C++-based inference system and profile wrapper, as shown in Fig. 3. We used a C++ inference implementation, specifically LlaMA.Cpp [18], due to the performance orientation of C++ algorithms and its relatively easy integration with hardware profilers. This stage has two modes: a performance mode, where Lumina only evaluates high-level metrics, and a profile mode where it generates a detailed profile analysis using the available hardware level profilers. Every time it process a new query, Lumina checks the execution mode. This allows to switch between modes in runtime without restarting Lumina.

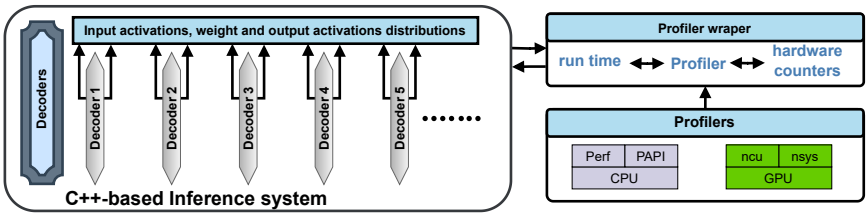

Fig. 3. Overview of the Execution-profiling stage.

A. LLM inference using LlaMA.Cpp. LlaMA.Cpp is an open-source C++ project designed to enable the efficient execution of large language models on diverse hardware platforms. It provides a lightweight, fast, and resource-efficient runtime. In Lumina, we adapt llama.cpp by fully removing its original frontend and utilizing it purely as a backend library. This modification enables direct consumption of input embeddings generated by the feature extraction module, which are passed seamlessly from the Python-based preprocessing pipeline.

Beyond adapting the input interface, we also extended llama.cpp to support internal checkpoints for collecting input activations, output activations, and weight distributions during inference. These checkpoints are inserted at key locations within the forward pass, enabling Lumina to capture detailed per-layer statistics essential for performance analysis, debugging, and model optimization.

In addition, we instrumented LlaMA.Cpp with custom hooks to activate hardware profilers such as perf, papi, nsys, and ncu without altering the core model logic. These hooks are automatically triggered based on the execution mode (performance or profile). Although this work relies on llama.cpp as the inference backend, the architecture of Lumina remains fully plug-and-play: alternative attention systems or inference engines can be integrated, as long as they support embedding ingestion and provide similar checkpoints for weight and activation analysis, and expose hooks for profiler integration.

B. Profiler wrapper. As previously mentioned, we extended Llama.Cpp with hooks to interconnect hardware profilers with different its internal components, such as the layers. To enable integrating Lumina with any desire hardware-level profiler, we create the *Profiler wrapper*, an extendable library that maps the hooks integrated in LlaMA.Cpp to the profiling calls of any desire profiler allowing for more customization in hardware resource monitoring. If a profiler different from the already available ones in Lumina is required, the *Profilers wrapper* can be easily extend to integrate it. This adaptability ensures Lumina can be used in various hardware environments and for different profiling requirements.

Currently, Lumina uses *perf* and *PAPI* for CPU profiling, and *nsys* (NVIDIA Nsight Systems) and *ncu* (NVIDIA Compute Utilities), the official NVIDIA profiling tools for GPUs.

3.4 Output-analysis components

Lumina can generate two different set of outputs as shown by Fig. 2: High-level and hardware-level metrics. The *output-analysis* stage includes two separate modules which analyze and generate more friendly outputs, such as plots and tables. The High-level results include (i) the LLM's answer(s) to the given user or benchmark query(ies) and (ii) the following metrics: query latency, hardware utilization and cost. The hardware-level results include different metrics such as instructions per-cycle, cache miss/hit ratio, must time consuming instructions, among others. Hardware-level results are constrained by the available hardware counters in the baseline hardware platform. All the results can be given per executed query or a summary of all the results (e.g., geomean).

4 Evaluation

In this section, we evaluate three representative multi-modal and mixed-modal models, LLaVA-Mistral-7B, Chameleon-7B, and Deepseek-Janus-Pro-7B using Lumina. Our evaluation focuses on two complementary aspects: functional benchmarking on standard benchmarks and hardware-level profiling. This combined analysis provides insights into both the effectiveness and computational characteristics of the evaluated models.

4.1 Hardware Platforms

We evaluate both CPU and GPU systems. Our CPU baseline consists of an Intel(R) Xeon(R) Platinum 8480+, a server class CPU with 56× 6-way hardware threads and 105MB of cache. For the GPU baseline, we use four NVIDIA H100.

4.2 Functional Benchmarking Methodology

We employed five widely recognized benchmarks covering a diverse range of multimodal and mixed-modal tasks. Table 1 summarizes the key characteristics of each evaluated benchmark.

Table 1. Benchmarks Details

Benchmark	Type	# Queries	# Categories	Description
MMBench-Dev [11]	Text+Image	4329	20	Perception and reasoning
MMLU [20]	Text	15908	57	Factual knowledge and reasoning
MMMU-Pro [23]	Text+Image	1730	30	Multi-disciplinary complex reasoning
OCRBench [6]	Image	1000	10	OCR-based understanding-reasoning
SEEDBench-IMG [8]	Image	14232	57	Spatial and commonsense reasoning

4.3 Profiling Methodology

We conducted a detailed profiling to analyze the hardware-level behavior of the models during inference. We profile both CPU-based and GPU-based inference implementations, and report the following metrics. For GPUs we report results using one and four GPUs.

- **CPU metrics:**
  - Memory instructions ratio
  - Instructions Per Cycle (IPC)
  - L2, and L3 cache miss rates
- **GPU metrics:**
  - Memory Bandwidth
  - Kernel Execution Time

The aforementioned hardware metrics were evaluated in this paper due to space limitations. However, all the profilers included in Lumina support a significant larger number of metrics that can be evaluated using our framework. The profiling process is seamlessly integrated into Lumina and automatically adapts to the benchmark configuration, ensuring non-intrusive and reproducible collection of hardware-level information.

Experiments All benchmarking and profiling experiments are conducted under the 4-bit quantization setting, enabled by our custom extension of VLMEvalKit to support low-bit inference. Additionally, our framework retains compatibility with full-precision execution.

4.4 Benchmarking Results

We now present the results of our functional evaluation, where we assess the effectiveness of each model on the benchmarks listed in Table 1. We report the accuracy, inference time and number of tokens generated for all the evaluated models accross all the benchmarks. In the following, we compare the models both quantitatively and qualitatively across benchmarks.

A. Model accuracy. Fig. 4 (left) depicts the accuracy results across all the experiments. The figure clearly shows that LLaVA-Mistral-7B consistently achieves the highest accuracy in the majority of benchmarks, especially excelling on the pure text benchmark MMLU. Deepseek-Janus-Pro-7B outperforms to LLaVA-Mistral-7B on OCRBench and MMMU-Pro. These benchmarks evaluate complex reasoning between text and images, where the unified architecture of Deepseek-Janus-Pro-7B provides better results. In contrast, Chameleon-7B consistently underperforms, with notably lower accuracy on benchmarks requiring complex reasoning such as MMBench-Dev, MMMU-Pro and OCRBench. These results highlight the superior generalization of LLaVA-Mistral-7B, the OCR-oriented capabilities of Deepseek-Janus-Pro-7B, and the limitations of Chameleon-7B across diverse benchmarks.

B. Inference time. Fig. 4 (right) reports the CPU inference times of the evaluated models. Deepseek-Janus-Pro-7B consistently achieves lower inference latency across most benchmarks, with Chameleon-7B showing moderate performance and LLaVA-Mistral-7B generally exhibiting the highest latency. As evaluated in the next section (token generation), the number of input tokens generated per model dictates its performance.

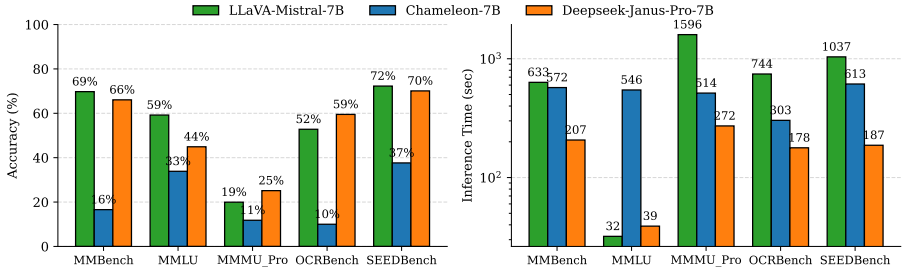

Fig. 4. Accuracy (left) and inference time (right) comparison of LLaVA-Mistral-7B, Deepseek-Janus-Pro-7B, and Chameleon-7B across five benchmarks.

C. Tokens generation. Fig. 5 reports the average number of input and output tokens processed per query. Notably, LLaVA-Mistral-7B tends to process significantly more input tokens, especially on MMBench-Dev, MMMU-Pro, and SEEDBench-IMG, which correlates with its longer inference times. On the other hand, Deepseek-Janus-Pro-7B consistently operates on fewer tokens, which contributes to its faster execution. For the output tokens, Chameleon-7B produces disproportionately long outputs, particularly in and MMBench-Dev. The main reason behind is that even when a single A, B or C option answer is requested, Chameleon-7B's answer always include an explanation. Even after carefully revising and modifying the queries in the benchmarks, this situation continued. In contrast, both LLaVA-Mistral-7B and Deepseek-Janus-Pro-7B produce shorter, more concise outputs, which may align better with benchmark requirements.

D. GPU inference time. Fig. 6 present the inference times on different number of GPUs. Deepseek-Janus-Pro-7B achieves the lowest latency across benchmarks, followed by LLaVA-Mistral-7B, while Chameleon-7B consistently incurs the highest inference times, especially on MMBench-Dev, MMLU, and SEEDBench-IMG. Scaling from 1× to 4× GPUs does not result in significant latency reductions for any of the models. This is primarily because even a single GPU that we used already provides sufficient computational capacity to execute the models without saturating available resources. Consequently, adding more GPUs does not lead to noticeable improvements under the batch size and workload used. These results suggest that inference latency in this setting is

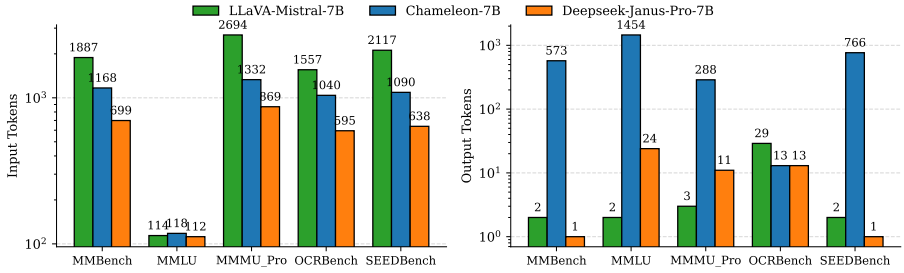

Fig. 5. Average number of input tokens (left) and output tokens (right) processed by LLaVA-Mistral-7B, Chameleon-7B, and Deepseek-Janus-Pro-7B.

dominated by the models' intrinsic architectural and token consumption characteristics, rather than GPU hardware limitations.

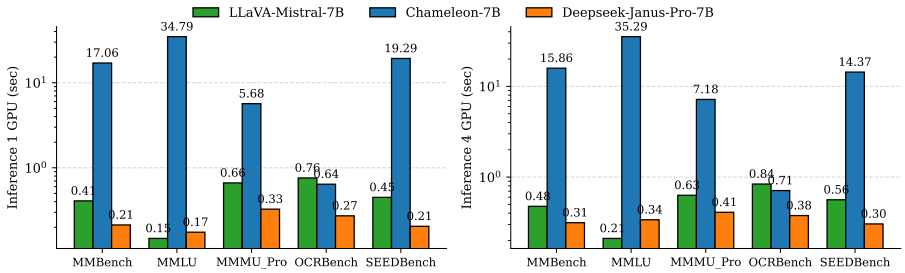

Fig. 6. Inference time comparison across benchmarks and models on 1×GPU (left) and 4×GPU (right).

4.5 CPU profiling results

We profile the models running on CPU backends using `PAPI` profiler. PAPIprovides a detailed view of the computational workload and memory subsystem behavior during inference.

A. Memory instructions ratio and Instructions per cycle. Fig. 7 reports the Memory instruction ratio (left) and Instructions per-cycle (right, IPC) for all the evaluated experiments. LLaVA-Mistral-7B features the higher memory instructions ratio, especially on MMMU-Pro, reflecting its higher input token usage. Conversely, Deepseek-Janus-Pro-7B demonstrates lower memory access intensity, consistent with its shorter token sequences and more efficient memory utilization.

All models feature comparable IPC values across benchmarks, with Deepseek-Janus-Pro-7B slightly outperforming others in most cases. The baseline CPU

includes 6-way Out-of-order cores, therefore, the IPC achieves is around 60% to 70% of the peak performance. This results can be attributed to resource contention due to the high number of memory instructions executed. It is noteworthy to mention how this results highlight new opportunities to further improve the performance of these model over the baseline CPU.

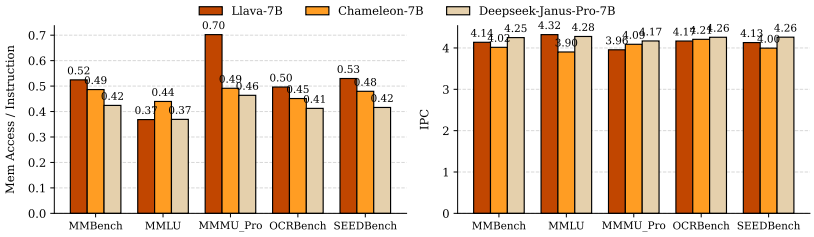

Fig. 7. Memory instructions ratio (left) and instructions per cycle (right).

B. Cache miss ratio. Fig. 8 shows that Chameleon-7B experiences significantly higher L2 and L3 cache miss rates, especially on MMLU and SEEDBench-IMG. This is likely due to its tendency to generate considerably longer outputs compared to Deepseek-Janus-Pro-7B and LLaVA-Mistral-7B, resulting in larger memory footprints. In contrast, the lower miss rates observed for Deepseek-Janus-Pro-7B and LLaVA-Mistral-7B can be attributed to their more compact output generation and moderate input sizes, leading to better cache utilization.

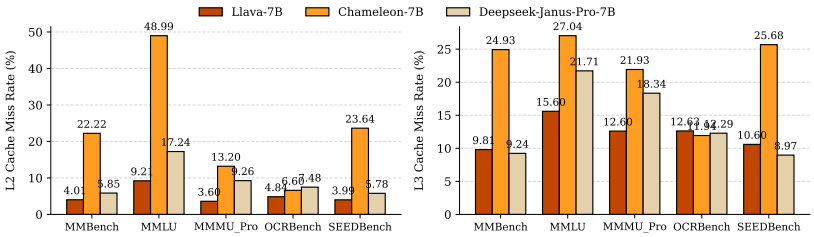

Fig. 8. L2 and L3 cache miss rate.

C. CPU take aways. Our profiling evaluation using Lumina demonstrates that there are still multiple areas to improve performance. For instance, the high number of memory instructions combined with the high number of cache misses depict that the cache hierarchy is underutilize and could be further improvements. This reflects in the 60% to 70% of the peak performance achieved by all the models. By further analyzing other hardware parameters using Lumina could bring light to other performance degration sources.

4.6 GPU profiling results

We profile the models running on GPU backends using `nsys` which provides insights into GPU kernel execution, memory operations, and hardware-level performance during inference.

A. Memory transfer analysis. Fig.9 illustrates that LLaVA-Mistral-7B and Deepseek-Janus-Pro-7B generally exhibit larger host-to-device memory transfer sizes compared to Chameleon-7B, especially noticeable on MMLU and SEEDBench-IMG. However, as shown in the right plot, both LLaVA-Mistral-7B and Deepseek-Janus-Pro-7B achieve lower or comparable transfer latencies than Chameleon-7B, despite handling larger data. This suggests that Chameleon-7B may suffer from fragmented or less efficient memory transfers, leading to a higher number of smaller copies, which is typically suboptimal for GPU memory operations. This inefficiency is also in line with the previously observed higher inference time of Chameleon-7B.

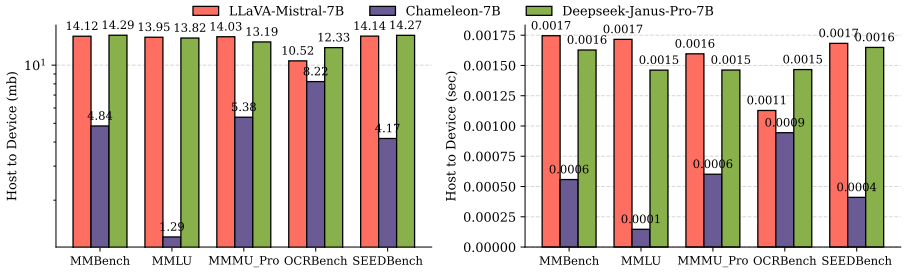

Fig. 9. Host-to-Device memory transfer analysis on 1 GPU: average transferred data size (left) and average transfer latency (right) across benchmarks and models. Larger bars in the left plot indicate higher average transfer size per operation, while the right plot shows the corresponding latency per operation.

Kernel utilization and Memory bandwidth. As illustrated in Fig.10, we compare the kernel utilization and memory bandwidth of three prominent vision-language models: LLaVA-Mistral-7B, Chameleon-7B, and Deepseek-Janus-Pro-7B across five benchmark tasks (10c, EN, IMG, MMLU, and OCRBench). Chameleon-7B consistently achieves the highest kernel utilization across all tasks, peaking at 94.0% on the MMLU benchmark. In contrast, LLaVA-Mistral-7B and Deepseek-Janus-Pro-7B show notably lower utilization, particularly on IMG and MMLU, where values drop to as low as 14.7% and 13.0%, respectively. Despite the differences in kernel usage, memory bandwidth remains relatively consistent across models. All three models maintain a bandwidth between approximately 8 MB/s and 10.4 MB/s, with Chameleon-7B reaching the highest value of 10.37 MB/s on the MMLU task. These findings highlight Chameleon-7B's superior GPU re-

source efficiency, suggesting potential advantages in scalability and performance under compute-constrained settings.

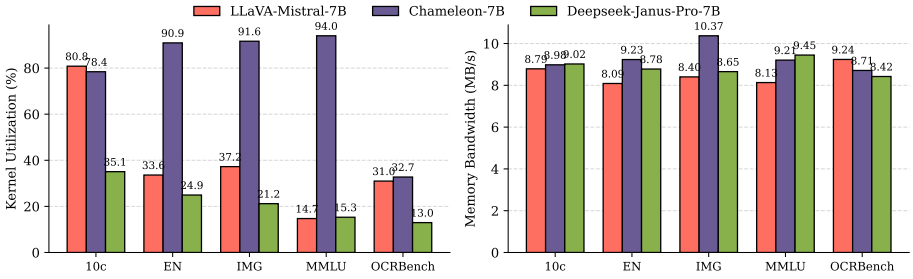

Fig. 10. Kernel utilization and memory bandwidth

4.7 Input, Weight and Output Distributions

In addition to benchmarking and profiling, we conduct a layer-wise analysis of the models by examining the statistical distributions of layers inputs, weights, and-outputs. For this purpose, we instrument the models with dedicated checkpoints to capture these distributions at every layer during inference. This analysis enables us to investigate how information flows through the model, layer by layer, revealing patterns related to representation dynamics, activation variability, and potential bottlenecks. The per-layer statistics provide valuable insights into the internal operation of the models beyond their final outputs.

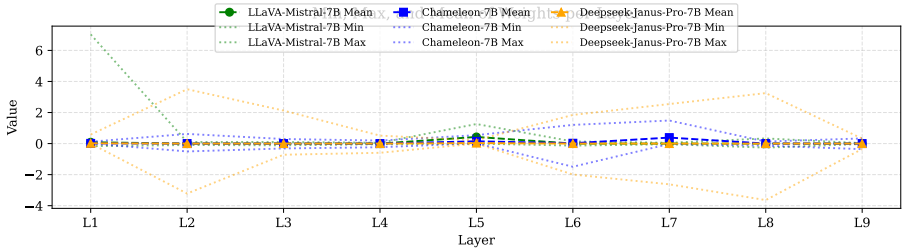

Fig. 11. Weight distributions

Fig.11 illustrates the minimum, maximum, and average weight values across nine layers for the LLaVA-Mistral-7B, Chameleon-7B, and Deepseek-Janus-Pro-7B models. This figure serves to demonstrate that the profiling tool is capable of capturing detailed statistics on a per-layer basis. The visualization highlights the granularity and richness of the collected data, emphasizing the tool's ability to extract low-level insights across all layers.

5 Conclusion and Future Work

In this work, we introduced Lumina, a unified and modular framework for performance evaluation and hardware profiling of Large Language Models (LLMs). By combining the flexibility of Python with the low-level control of C++, Lumina enables fine-grained analysis across both multi-modal and mixed-modal architectures on diverse hardware platforms. Through extensive profiling of LLaVA-Mistral-7B, Chameleon-7B, and Deepseek-Janus-Pro-7B, we demonstrated the framework's capability to expose hardware-level bottlenecks and reveal differences in computational efficiency.

Our experiments show that Deepseek-Janus-Pro-7B significantly outperforms the other evaluated models in both execution time and overall efficiency. Additionally, we presented the first OCR-based evaluation of Chameleon-7B and Deepseek-Janus-Pro-7B, providing insights into their document understanding capabilities under constrained input resolutions.

Looking ahead, we plan to extend Lumina in two major directions. First, we aim to develop a Large Language Model capable of interpreting profiling outputs and producing actionable insights to guide hardware-aware optimization. This model would serve as an intelligent assistant for understanding low-level performance data and suggesting architecture-specific improvements. Second, based on the collected profiling data, we will focus on identifying the most time-consuming instructions or operations within LLM execution and applying targeted acceleration techniques. This includes optimizing critical paths using instruction-level parallelism, cache-aware scheduling, and quantization strategies for both CPUs and GPUs. These future enhancements will enable Lumina to move beyond passive profiling and towards active performance optimization, offering a more intelligent and adaptive approach for evaluating and improving next-generation LLMs.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: {TensorFlow}: a system for {Large-Scale} machine learning. In: 12th USENIX symposium on operating systems design and implementation (OSDI 16). pp. 265–283 (2016)
2. AI, D.: Janus-pro: Deeply coupled vision-language models. In: arXiv preprint arXiv:2401.05538 (2024)
3. Duan, H., Yang, J., Qiao, Y., Fang, X., Chen, L., Liu, Y., Dong, X., Zang, Y., Zhang, P., Wang, J., et al.: Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. In: Proceedings of the 32nd ACM International Conference on Multimedia. pp. 11198–11201 (2024)
4. Face, H.: Transformers: State-of-the-art machine learning for pytorch, tensorflow, and jax. https://huggingface.co/docs/transformers/index (2024)
5. Frederickson, B.: Py-spy: Sampling python profiler. https://github.com/benfred/py-spy (2024)
6. Fu, L., Yang, B., Kuang, Z., Song, J., Li, Y., Zhu, L., Luo, Q., Wang, X., Lu, H., Huang, M., Li, Z., Tang, G., Shan, B., Lin, C., Liu, Q., Wu, B., Feng, H., Liu, H.,

Huang, C., Tang, J., Chen, W., Jin, L., Liu, Y., Bai, X.: Ocrbench v2: An improved benchmark for evaluating large multimodal models on visual text localization and reasoning (2024), https://arxiv.org/abs/2501.00321
7. Innovative Computing Laboratory: PAPI: Performance Application Programming Interface (2025), https://icl.utk.edu/papi/, accessed: 2025-04-03
8. Li, B., Wang, R., Wang, G., Ge, Y., Ge, Y., Shan, Y.: Seed-bench: Benchmarking multimodal llms with generative comprehension (2023), https://arxiv.org/abs/2307.16125
9. Li, J., Li, D., Savarese, S., Hoi, S.: Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In: International conference on machine learning. pp. 19730–19742. PMLR (2023)
10. Liu, H., Li, C., Li, Y., Lee, Y.J.: Improved baselines with visual instruction tuning (2024), https://arxiv.org/abs/2310.03744
11. Liu, Y., Duan, H., Zhang, Y., Li, B., Zhang, S., Zhao, W., Yuan, Y., Wang, J., He, C., Liu, Z., Chen, K., Lin, D.: Mmbench: Is your multi-modal model an all-around player? In: Leonardis, A., Ricci, E., Roth, S., Russakovsky, O., Sattler, T., Varol, G. (eds.) Computer Vision – ECCV 2024. pp. 216–233. Springer Nature Switzerland, Cham (2025)
12. NVIDIA Corporation: Nvidia nsight systems. https://developer.nvidia.com/nsight-systems (2025), accessed: 2025-04-03
13. Perf Community: Perf wiki - linux performance analysis tools. https://perfwiki.github.io/main/ (2025), accessed: 2025-04-03
14. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI blog **1**(8), 9 (2019)
15. Radford, A., et al.: Learning transferable visual models from natural language supervision. In: Proceedings of ICML. pp. 8748–8763 (2021)
16. Team, C.: Chameleon: Mixed-modal early-fusion foundation models. arXiv preprint arXiv:2405.09818 (2024)
17. Team, P.: Pytorch profiler. https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html (2024)
18. Team, T.L.: Llama.cpp: A high-performance c++ inference for llama models. https://github.com/ggerganov/llama.cpp (2024)
19. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2023), https://arxiv.org/abs/1706.03762
20. Wang, Y., Ma, X., Zhang, G., Ni, Y., Chandra, A., Guo, S., Ren, W., Arulraj, A., He, X., Jiang, Z., Li, T., Ku, M., Wang, K., Zhuang, A., Fan, R., Yue, X., Chen, W.: MMLU-pro: A more robust and challenging multi-task language understanding benchmark. In: The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2024), https://openreview.net/forum?id=y10DM6R2r3
21. Weaver, V.M., et al.: Performance counters: a survey of hardware monitoring and analysis tools. ACM Computing Surveys (2013)
22. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations. pp. 38–45 (2020)
23. Yue, X., Zheng, T., Ni, Y., Wang, Y., Zhang, K., Tong, S., Sun, Y., Yu, B., Zhang, G., Sun, H., Su, Y., Chen, W., Neubig, G.: Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark (2024), https://arxiv.org/abs/2409.02813