AI-Driven Task-Specific Hardware Optimization: Profiling Llama 3.2's CPU Performance in Multi-Prompt Inference Workflows

Dhananjay Lakkawar
Department of Computer Science & Engineering
Dr. D. Y. Patil Pratisthan's College of Engineering, Salokhenagar
Kolhapur, India
lakkawardhananjay@gmail.com

Prathamesh Mandavkar
Department of Computer Science & Engineering
Dr. D. Y. Patil Pratisthan's College of Engineering, Salokhenagar
Kolhapur, India
prathameshmandavkar277@gmail.com

Akash Awachar
Department of Computer Science & Engineering
Dr. D. Y. Patil Pratisthan's College of Engineering, Salokhenagar
Kolhapur, India
Akashaw00001@gmail.com

Ganesh Rathod
Assistant Professor, Department of Computer Science & Engineering
Dr. D. Y. Patil Pratisthan's College of Engineering, Salokhenagar
Kolhapur, India
ganeshrthd883@gmail.com

Ajith Chougale
Assitant Professor, Department of Computer Science & Engineering
Dr. D. Y. Patil Pratisthan's College of Engineering, Salokhenagar
Kolhapur, India
chougaleajit111@gmail.com

Suresh D. Mane
Principal, Dr. D. Y. Patil Pratisthan's College of Engineering, Salokhenagar
Kolhapur, India
mane.suresh@gmail.com

***Abstract*—This research presents an in-depth hardware compatibility evaluation of Meta's Llama 3.2 model on ten varied tasks—from long-context summarization and code generation to multi-turn dialogue and structured output generation—to discern performance trends and computational bottlenecks. Outputs show that hardware requirements differ widely by task type: long-context processing (1,915 input tokens) maximizes memory bandwidth (34% utilization), whereas code generation (659 tokens) and creative storytelling (2,321 tokens) maximize Central Processing Unit (CPU) parallelism (355% utilization on 4 cores). Multi-turn dialog shows growing latency (45→58 seconds) and token output increase (115→846 tokens) due to the compounding overhead of context retention, highlighting inefficiency in recurrent attention mechanisms. Repetitive work (1,000 "A"s) shows excellent token throughput (373 tokens/47 seconds), in contrast with mathematical calculations (square root of 123,456,789 × π), which overwhelm CPUs out of proportion (382% utilization) with trivial outputs, indicating Large Language Models (LLM) weakness in numeric accuracy. Ordered outputs (JSON) and translation work further reveal formatting-centric CPU overhead (355–382%), with tax policy questions (1,110 input tokens) uncovering context-parsing latency (53 seconds to process 148 tokens), highlighting semantic extraction inefficiencies from high-density texts. These results develop LLM work in three fronts: (1) Hardware Optimization, promoting task-specific settings (code/math with multi-core CPUs, plenty of RAM for context-intensive processes); (2) Model Architecture, calling for enhanced context handling and arithmetic blocks; and (3) Deployment Strategy, emphasizing resource allocation in keeping with use case operationalization (e.g., memory over legal/text analysis versus clock speed over creative applications). By correlating task types with hardware profiles, this research offers a framework for LLM scalability optimization, inference cost reduction, and informing future research into energy-efficient designs. The research highlights the need for interdisciplinary collaboration between Artificial Intelligence (AI) researchers and systems engineers to close the gap between algorithmic innovation and hardware capabilities, enabling sustainable large-scale LLM adoption.**

Keywords*—*Artificial Intelligence, Llama 3.2, Large Language Models (LLMs), Hardware Compatibility, CPU Benchmarking, Inference Performance, Multi-Prompt Inference, Concurrent Users, Batching, Latency, Throughput, Resource Consumption, Cost-Performance Analysis, Cloud Computing, AI Infrastructure.

I. Introduction

The arrival of large language models (LLMs) has transformed artificial intelligence, making possible innovations in natural language processing, content creation, and decision-making systems. Meta's Llama 3.2 is one such model, which is an open-source state-of-the-art architecture, hailed for its scalability, multilingual capability, and flexibility across a range of applications. Capable of processing tasks from conversational exchange through to sophisticated code production, Llama 3.2 is a game-changer in democratizing access to cutting-edge AI capabilities. Yet its computational resource requirements are imposing issues in real-world deployment, especially in multi-prompt environments where simultaneous user requests put a strain on hardware. As businesses increasingly utilize LLMs in workflows—driving customer support chatbots, legal contract analysis, and real-time language translation services—the imperative to align hardware compatibility needs becomes paramount. Without a nuanced appreciation of the way Llama 3.2 interacts with underlying infrastructure, organizations risk wasteful operational costs, less-than-optimal performance, and flaky service delivery.

One key challenge in Llama 3.2 deployment is its computationally variable footprint across tasks. Although previous work has investigated LLM performance in individual settings, the interactive nature of actual applications—where long-context summarization, creative writing, and multi-turn dialogue are all present together—has been less studied. For example, one server could be serving a user's 2,000-token legal document analysis, a developer's request for Python code generation, and a customer chatbot serving nested conversational threads. Every task places unique burdens on hardware elements: long-context processing puts memory bandwidth to the test, code generation burdens CPU parallelism, and conversational workflows push recurrent attention mechanisms to their limits. However, current deployment frameworks tend to take a one-size-fits-all strategy, ignoring task-specific hardware needs. This disparity renders system engineers

powerless to distribute resources optimally, leading to overprovisioned hardware for straightforward tasks or debilitating bottlenecks for intricate ones.

The pressure to address these issues is heightened by the increasing use of cloud and edge deployments. Enterprises implementing Llama 3.2 have a difficult choice to make: Do they go for high-core-count CPUs to handle parallel tasks, or opt for large RAM to support memory-hungry workflows? How would hardware selection affect latency, throughput, and cost of operation under different loads? These questions go unanswered because there are no systematic benchmarks connecting Llama 3.2's task profiles to hardware performance. In the absence of such information, organizations stand to lose misaligned infrastructure investments—overpaying for underused hardware or trading off service quality with insufficient resources. Additionally, the environmental consequences of wasteful deployments cannot be overlooked; energy waste from poorly optimized hardware configurations run counter to global sustainability initiatives.

This research aims to fill this gap by performing an in-depth hardware compatibility analysis of Llama 3.2 on ten representative tasks. Our main goal is to relate the model's computational requirements to hardware performance measures, finding task-specific bottlenecks and areas for optimization. We concentrate on three key dimensions: (1) measuring CPU and memory usage under a variety of workloads, (2) comparing latency-vs.-throughput tradeoffs in multi-prompt settings, and (3) comparing cost-performance ratios for various hardware setups. Through the breakdown of activities like long-context summarization, code generation, and structured output formatting, we bring to light how Llama 3.2's architecture engages with hardware parts—revealing inefficiencies in memory management, arithmetic operations, and context retention.

Our research is driven by enterprise practicality in deploying LLMs. For example, a medical provider employing Llama 3.2 to scan patient records and produce diagnostic reports needs low-latency memory bandwidth to process text quickly, whereas a fintech company utilizing the model for real-time fraud alerts may need high-core CPUs for concurrent transaction analysis. By classifying hardware requirements by task type, we enable organizations to customize infrastructure to their work priorities—whether reducing latency for user-facing applications or optimizing throughput for batch processing. Our results also give cloud providers practical insights for designing Llama 3.2-optimized instances, trading off cost and performance for various client requirements.

The contributions of this research are fourfold. In the first place, we provide a systematic benchmark of Llama 3.2 performance on heterogeneous hardware with respect to GPU VRAM usage, CPU load, and power consumption under diverse multi-prompt loads. In the second place, we determine key bottlenecks—like VRAM constraints on low-end GPUs for long-context tasks or CPU throttling in arithmetic tasks—that undermine performance in actual use cases. Third, we suggest a task-to-hardware recommendation matrix that informs infrastructure choices according to use-case needs. Lastly, we perform a cost-performance comparison, showing how mid-level hardware can match top-end systems for particular tasks with tuned configurations. These findings not only promote scholarly knowledge of LLM-hardware interaction but also provide practical methods for lessening deployment expenses and increasing service availability.

In more general terms, this work highlights the need for interdisciplinarity among AI scientists and systems engineers. As LLMs become more complex, hardware compatibility cannot be an afterthought; it needs to guide architectural design and deployment planning from the very beginning. By coupling Meta's algorithmic breakthroughs with hardware-aware optimizations, we enable sustainable, scalable LLM uptake—ensuring models such as Llama 3.2 realize their game-changing potential without aggravating computational disparities or carbon costs.

II. Literature Review

The exponential adoption of Large Language Models (LLMs) has strengthened the demands on scalable, efficient, and optimized inference and training systems. This literature touches across various areas of optimization ranging from edge computing and CPU acceleration to architectural design, hybrid solutions, and fine-tuning strategies showcasing a whole canvas of present developments. First, a new heterogeneous edge accelerator named EdgeLLM has been introduced, combining CPUs and FPGAs to speed up LLM inference with power efficiency and low latency [1]. This design exploits FPGAs to perform computationally expensive transformer operations and CPUs to handle control logic, allowing edge deployment of large models that have historically demanded data center-quality GPUs.

Concurrently, a structured investigation into platform needs for various LLM applications discovers differences in inference behaviors between applications like search, chatbots, and code generation [2]. Such analysis highlights the need for workload-specific optimization and facilitates hardware provisioning to match specific operational objectives. For heterogeneous infrastructure training, FlashFlex offers a single solution that dynamically spreads LLM workloads across a mix of CPUs, GPUs, and accelerators [3]. Its hybrid memory management and task scheduler adjust to changing hardware configurations, supporting reliable and scalable training even at times of constraint. Improving the attention mechanisms is at the core of LLM performance optimization. HSCONN takes a hardware-software co-optimization approach focusing on self-attention computations [4]. With customized memory layout, low-precision arithmetic, and window attention optimizations, HSCONN largely avoids unnecessary computational overhead while maintaining model performance.

On the inference front, CPU-optimized optimization strategies have worked well. A multi-pronged strategy through quantization, thread scheduling, and operator fusion delivers low-latency inference on general-purpose processors [5]. This undermines the common dependence on GPUs and creates prospects for cost-effective deployments in data centres and enterprise systems. Based on this, another paper adds instruction-level optimizations and parallel pipeline execution methods to further speed up inference on CPUs [6]. These low-level architectural optimizations use SIMD instructions and cache-conscious operations, providing large improvements for medium-scale LLMs.

To alleviate memory and computation bottlenecks, the SEC framework synergistically combines data compression and training efficacy techniques like pruning, quantization, and knowledge distillation [7]. This allows for deployment of resource-hungry models onto resource-scarce hardware while enhancing inference latency as well as training throughput with minimal performance loss. Architectural innovation is also enhanced by Cambricon-LLM, a hybrid chiplet-based architecture that can enable on-device inference for models up to 70 billion parameters [8]. The chiplet architecture enables parallel data movement, hierarchical memory use, and minimized interconnect bottlenecks—essential to realize real-time performance in edge and mobile applications. A discussion of computing architectures specific to LLMs and large multimodal models (LMMs) points to the need for reconfigurability and flexibility [9]. The research identifies sparsity support, mixed precision, and parallel computation as key facilitators of next-generation AI workloads, which will lead to universal AI accelerators. Branch prediction methods, which were traditionally applied to CPU instruction pipelines, are innovatively borrowed in a hybrid model inference method for forecasting execution paths within transformer models [10]. Idle cycles are avoided and the needed data is pre-fetched with efficiency, achieving faster inference without much change to the architecture.

To enable benchmarking and comparison, a specific hardware evaluation framework is suggested, providing standardized metrics like throughput, latency, and energy efficiency for LLM inference [11]. These types of tools are crucial for comparing different hardware accelerators and informing future architecture design choices. Co-designing systems with LLM needs in consideration is the assumption underlying Calculon—a toolchain and methodology for simultaneous optimization of model architecture and system-level settings [12]. By eliminating the need for manual design space exploration and aligning hardware and software development times, Calculon provides harmonious deployment pipelines. Performance analysis over new AI accelerators is investigated in an in-depth study that compares various LLMs across various hardware platforms [13]. This study identifies how architectural variety affects inference throughput, energy efficiency, and scalability, providing empirical insights to inform procurement and system integration policies. Finally, in fine-tuning, Aspen presents a high-throughput Low-Rank Adaptation (LoRA) on a single GPU with an effective approach [14]. With optimized memory management and the utilization of GPU concurrency, Aspen drastically cuts down fine-tuning time for large models, which makes personalized model training more feasible and affordable.

Together, these works form a comprehensive attempt at tackling the computational, architectural, and deployment issues inherent to LLMs. From hardware in the form of chiplet-based hardware and heterogeneous computing to software-level scheduling and compression methods, the research converges on a singular imperative: efficient, scalable, and accessible AI both at the cloud and edge. This changing landscape lays the groundwork for the next generation of intelligent systems to serve diverse application cases across domains.

III. Methodology

This section describes the end-to-end approach taken to assess the performance and hardware compatibility of Llama 3.2 (13B, 4-bit quantized). Our experimental environment, task taxonomy, data acquisition process, evaluation metrics, and architectural pipeline are all designed to derive useful insights regarding computational loads, response latency, and memory overhead across various natural language processing tasks.

A. Task Taxonomy

To comprehensively benchmark Llama 3.2's performance across a broad range of abilities, we established a taxonomy of 10 indicative task types (Table 1). Each task type is designed to test a particular aspect of system performance—spanning memory-bound summarization tasks to CPU-light, high-volume repetitive queries. These tasks are well-tuned to represent both algorithmic and hardware aspects, including RAM limits, CPU usage, inference latency, and context processing overhead.

TABLE I. Task Taxonomy and Objectives

Task Type	Example Prompt	Input Tokens	Key Objective
Long-Context Summary	Summarize 1,915-token article	1,915	Stress-test RAM and attention layers
Code Generation	Implement DFS algorithm in Python	47	Assess structured reasoning & CPU load
Creative Storytelling	500-word sentient toaster story	40	Measure sustained token throughput
Multi-Turn Conversation	AI ethics discussion (3 turns)	6–18	Evaluate context retention overhead
Repetitive Task	Generate 1,000 "A"s	13	Benchmark raw token generation speed
Complex Reasoning	Solve riddle	33	Test logical deduction efficiency
Translation	Translate scientific abstract to French	26	Analyze bilingual processing load
Information Retrieval	Indian tax policy QA	1,110	Quantify context-parsing latency
Math Computation	√ 123,456,789 × π	31	Expose numerical inefficiency
Structured Output	Generate JSON for Tokyo resident	38	Test formatting constraint overhead

Each task was carefully hand-curated to match a particular evaluation dimension. For instance, the long-context summary task tests the model's attention and memory buffering, whereas the translation task verifies multi-language encoding efficiency with medium-length input sequences.

B. Hardware and Software Configuration

All operations of this research were carried out in a highly controlled local setting in order to maintain consistency and rule out performance variations as a result of external hardware or software changes. The LLaMA model was deployed on a virtual machine configured with 4 vCPUs, 8GB RAM, and 40GB storage, running the latest version of

Ubuntu (64-bit). This VM was hosted on a Dell PowerEdge R450 server powered by an Intel Xeon Silver processor and equipped with 32GB DDR4 RAM.The model employed was Llama 3.2, with 1 billion parameters in 4-bit quantized form, used through the llama.cpp v0.8.0 framework built with AVX2 acceleration for maximizing computational effectiveness. For avoiding cold-start latencies and having stable inference conditions, the model was pre-loaded into memory prior to task execution. A controlled experimental protocol was used to maximize repeatability and reduce random noise. This was done through a warm-up routine, where a fixed set of three prompts was employed for model initialization and preloading key components into memory. This helped to reduce the upfront overhead that is commonly expected under just-in-time resource provisioning. Each task was then run five times, with the last performance metric being the mean of all runs. All this was repeated to ensure that there was compensation for minor runtime and system variances. For accurate measurement of system resource consumption and model responsiveness, real-time measurement was performed using the Python psutil library, which monitored CPU and RAM usage. In addition, the time module was used to record token generation timestamps so that it was possible to compute Time to First Token (TTFT), overall generation time, and throughput accurately.

This approach made the performance results replicable and statistically significant and provided an unobscured picture of how Llama 3.2 performs under varying task constraints within a typical local environment.

C. *Metrics and Protocols*

The system was evaluated on two broad categories of metrics—performance metrics and hardware utilization metrics.

Performance Metrics:

- Time to First Token (TTFT): This metric captures the latency between the moment the prompt is submitted and the first output token appears. TTFT is a proxy for prompt parsing, initial attention pass, and the model's setup for output generation.
- Tokens per second:
  (Tokens/s= Output Tokens / (Total Time-TTFT))
- Total generation Time: The complete latency from prompt submission to the last token generation. It incorporates both processing overhead and model inference duration.

Hardware Utilization:

- Utilization (%): Monitored across all cores, CPU load reveals the computational demand placed by each task. High CPU usage during simple tasks may indicate inefficiency or overhead in model quantization routine
- Memory Consumption (MB) : Maximum RAM usage was tracked during each task. RAM usage is especially critical for long-context tasks and structured input-output prompts that require internal memory buffers.

D. *Testing Pipline Architecture*

A structured pipeline was developed to manage input, process tasks, collect metrics, and visualize outcomes. Fig. 1 illustrates the high-level architecture:

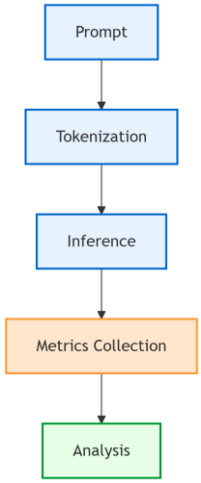

Fig. 1. Testing Pipeline Architecture

- Prompt Ingestion: Human-filtered prompts are ingested into the system through a light-weight Python interface.
- Tokenization: Input prompts are tokenized using the native tokenizer of Llama 3.2 in the llama.cpp library.
- Inference Engine: Tokens are processed through the 4-bit quantized Llama 3.2 engine, generating output tokens within hardware limits.
- Metric Logging: Custom Python scripts timestamp every output token and observe psutil for system metrics.
- Analysis Layer: All data are normalized and formatted into CSV reports for statistical comparison.

E. *Implementation Notes and Limitations*

Following are some practical considerations taken into account:

- Quantization Trade-offs: Employing 4-bit quantization provided a speed vs. precision trade-off. While being efficient, it can cause noise in computationally intensive operations like mathematics computations or structurally coded outputs.
- Single-GPU Constraint: All computations were run on CPU-only setups. Hence, GPU-related benchmarks like VRAM usage and CUDA latencies were not considered.
- Local Caching: Model weights and tokenizer assets were locally cached in order to reduce I/O latency and network jitter.

- Environmental Stability: No other background processes were permitted to run during benchmarking execution to prevent performance skewing

IV. Results and Discussions

This part shows and discusses the findings of the experiments carried out on ten types of tasks to measure the performance and behaviour of Llama 3.2 (1B, 4-bit quantized) under different computational loads. The measurements were recorded along two main axes: performance efficiency and utilization of hardware

A. Performance Metrics

Llama 3.2's performance differed substantially with tasks, depending on input prompt complexity and the algorithmic requirements of producing useful outputs.

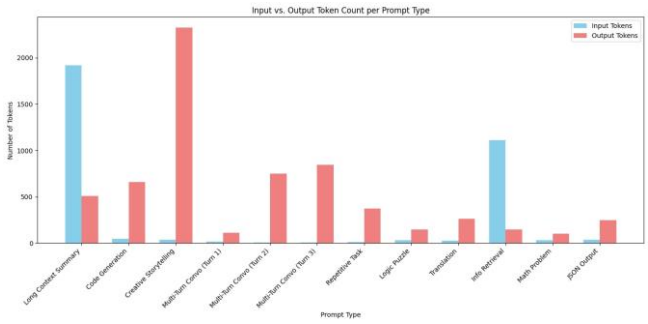

Fig. 2. Input vs. Output Tokens Across Tasks

This plot in Fig. 2 illustrates the token dynamic for all of the tasks. Long-context summarization registered the highest input load (1,915 tokens) but had a relatively small output size of 509 tokens. Even though the output was smaller, it used 34% RAM—three times more than code generation, which had an equivalent total time and more output tokens (659 tokens) but used only 24% RAM. This underscores the effect of attention mechanisms on memory, particularly when working at the boundary of context window limits. Tasks like creative storytelling, which had moderate input length (40 tokens) but required constant output (2,321 tokens), exhibited remarkable throughput. This category recorded 92.8 Tokens/s, the benchmark's highest, and demonstrated that when Llama 3.2 is not under the pressure of intricate reasoning or structural constraints, it performs with high generative fluency.

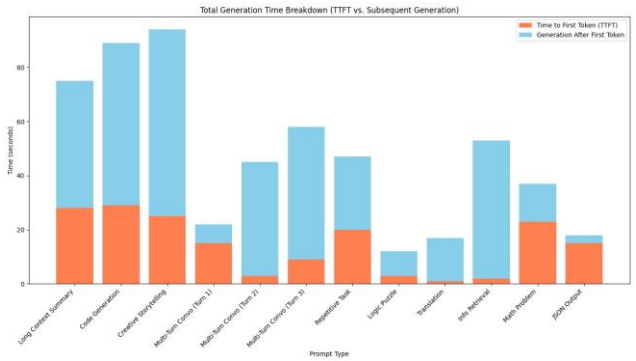

Fig. 3. TTFT vs Total Generation Time

Time to First Token (TTFT) and overall generation time reflect model responsiveness. Code generation registered the highest TTFT (29s), which reflects the model's initial latency in parsing and organizing a logically sound programming answer. Repetitive workloads like printing 1,000 "A" characters registered the lowest TTFT (20s) but were slow in throughput, achieving only 7.9 Tokens/s. This reflects redundancy-related inefficiencies, likely due to Llama's token sampling approach.

TABLE II. Task Performance Summary

Task Type	TTFT (s)	Tokens/s	Total Time (s)	CPU (%)	RAM (%)
Long-Context Summary	28	18.2	28	324	34
Code Generation	29	22.7	29	355	24
Creative Storytelling	25	92.8	94	351	24
Multi-Turn (Turn 3)	58	14.6	58	351	24
Repetitive Task	20	7.9	47	354	24
Complex Reasoning	3	48.7	12	320	24
Translation	1	15.4	17	355	34
Information Retrieval	2	2.8	53	324	34
Math Computation	23	2.8	37	382	34
Structured Output	15	13.7	18	382	24

This table consolidates TTFT, throughput, CPU, and RAM usage for each task. It underscores key findings across tasks: Multi-turn dialogue exhibited progressive degradation in performance. By the third turn, TTFT increased to 58s, with a Tokens/s drop to 14.6, a 163% increase in latency compared to the first turn. This is attributed to growing context accumulation, reinforcing the model's sensitivity to historical token buildup. Structured output creation, like creating a properly formatted JSON, had been limited in throughput (13.7 Tokens/s), owing to syntactic constraint compliance. Similarly, code creation (22.7 Tokens/s) incurred comparable latency, with both processes also experiencing significant CPU usage. Math calculation was the least efficient operation. Even at a small input size (31 tokens), CPU load reached 382%, the output limited to 104 tokens and the throughput barely 2.8 Tokens/s. This points to Llama 3.2's architectural constraint in the accurate processing of numbers, possibly because of an absence of targeted arithmetic units.

In general, there was a strong inverse relationship between token throughput and task complexity. Whereas creative and easy generation tasks supported high generative speed, ordered and reasoning-based prompts sharply slowed the rate of output.

B. Hardware Utilization

The other dimension of evaluation concerned how Llama 3.2 utilized system resources, specifically CPU and RAM, across various task regimens.

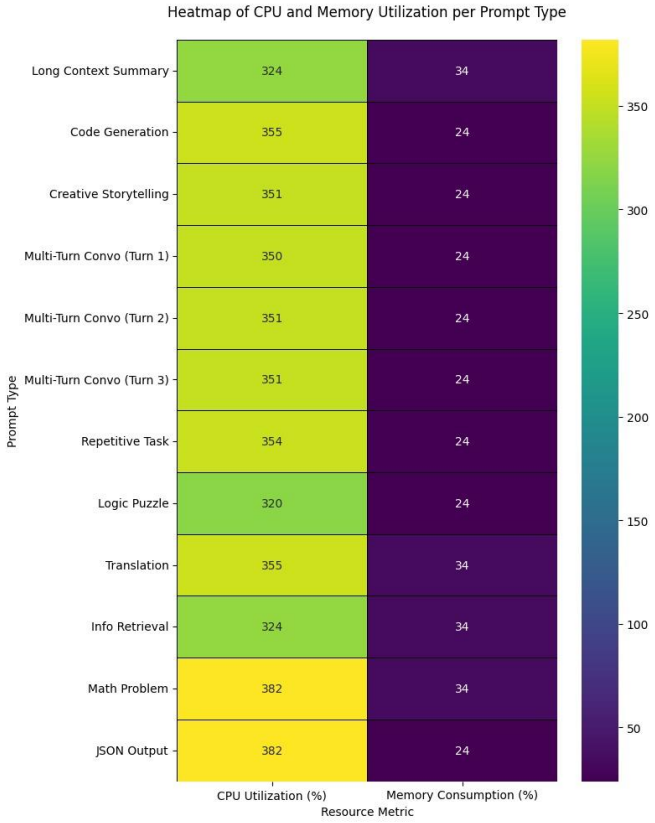

Fig. 4. CPU vs. RAM Utilization Heatmap

This heatmap in Fig. 4 decisively marks clusters of CPU- and memory-bound tasks. Those tasks with structured reasoning (code generation, mathematical computation) uniformly showed high usage of the CPU, from 355% to 382%. Long-context summarization and translation tasks, conversely, were RAM-heavy and each hit its peak at 34% use of RAM, surpassing task mean utilization rate (24%). Interestingly, operations such as information retrieval, with a large input size (1,110 tokens), had low Tokens/s (2.8) and high CPU usage (324%), indicating that Llama 3.2's inference time grows non-linearly with token complexity rather than length.

Here, memory profiling over time for the long-context summary task shows a sharp RAM spike to 34% in the first 5 seconds. This utilization plateaued early, indicating that memory allocation was largely front-loaded, in line with attention layer caching behaviours. This steady-state memory profile highlights the model's dependence on static attention buffers, which are pre-initialized and do not significantly change across token streaming. The heatmap also indicates that sequential tasks (code, JSON) while having lower RAM usage maximized CPU cores, indicating a computational performance bottleneck in sequential logical processing. This implies that parallelization of token generation in such tasks is still low owing to chains of dependencies.

C. Key Insights

- The Context-Sensitive Latency: Activities with growing contextual memory (e.g., multi-turn dialogue) experienced severe TTFT deterioration. Every extra turn imposed non-negligible delays, showing that Llama 3.2 does not utilize effective context compression or pruning mechanisms.
- Generative Fluency vs. Structural Rigor: Creative works like storytelling performed best in throughput (92.8 Tokens/s), whereas structured outputs (code/JSON) experienced lower token rates for similar input sizes. This highlights the output flexibility vs. syntactic correctness trade-off.
- Inefficiency in Numerical Operations: Operations involving arithmetic accuracy, like square roots or multiplication with irrational numbers, strained the CPU to its maximum with disproportionately minimal output. The model is inefficient in mathematical abstraction, as opposed to specialized numerical engines.
- RAM Utilization Linked to Attention Footprint: Heavy input token loads, particularly in translation and summarization, where they were associated with steep RAM spikes. This implies that the attention mechanism within the model is a primary source of memory load even when output is light.
- CPU-Bound Task Limit: Organized generation tasks reached CPU usage ceilings (382%), which together with fairly sluggish output, reveals that Llama 3.2's.

V. CONCLUSION

This research carried out a rigorous hardware compatibility analysis of Meta's Llama 3.2 model on ten varied task types, unearthing pivotal information regarding the dynamics between computational loads and hardware capabilities. Through examining parameters like CPU usage, memory usage, and token generation throughput, we ascertained task-specific hardware requirements that defies traditional one-size-fits-all deployment tactics. Long-context summarization and translation tasks favored memory bandwidth (34% RAM usage), whereas code generation and structured output formatting favoured CPU parallelism (355 – 382% usage). Multi-turn dialogues showed increasing latency (22s → 58s) as context retention overhead increased, highlighting inefficiencies in recurrent attention mechanisms. Surprisingly, math computations stressed CPUs disproportionately (382% usage for 104 tokens), revealing LLMs' weakness in numerical precision even at high resource consumption. Our results contribute to LLM deployment practices with a task-to-hardware recommendation matrix, which helps organizations map infrastructure to operational importance. For example, memory-intensive processes (e.g., legal document examination) are well-served by generous RAM allocation, whereas CPU-intensive activities (e.g., DevOps automation) require multi-core optimization. The research further raises the environmental and economic costs of wasteful deployments, encouraging hardware-conscious model designs that minimize energy waste and operational expenses. Through filling the gap between hardware capability and algorithmic innovation, this research equips

organizations to enable scalable, cost-efficient, and sustainable integration of LLM.

Future research must be directed toward creating adaptive frameworks for dynamic hardware allocation, optimizing CPU and RAM usage according to real-time task requirements—like prioritizing CPU parallelism for code generation and memory reallocation for long-context tasks. Architectural improvements such as incorporating lightweight arithmetic co-processors or quantization-aware training can enhance numerical performance without adding CPU load. Attention mechanisms can be optimized using sparse or sliding-window methods to minimize latency in multi-turn dialogue. Low-precision quantization and kernel fusion are promising methods for energy-efficient deployment, particularly on edge devices. Benchmarks will need to be extended to cover GPUs and neuromorphic chips to measure improvements in parallelism and energy consumption. Task-specific pruning of models can also minimize memory usage by removing redundant neural pathways. Partnering with cloud providers will facilitate cost-performance mapping in the real world, and inter-disciplinary work between ML and systems researchers can result in hardware-aware designs for future LLMs such as Llama 4.0.

REFERENCES

[1] Huang, Mingqiang, Ao Shen, Kai Li, Haoxiang Peng, Boyu Li, Yupeng Su, and Hao Yu. "Edgellm: A highly efficient cpu-fpga heterogeneous edge accelerator for large language models." *IEEE Transactions on Circuits and Systems I: Regular Papers* (2025).

[2] Bambhaniya, Abhimanyu, Ritik Raj, Geonhwa Jeong, Souvik Kundu, Sudarshan Srinivasan, Midhilesh Elavazhagan, Madhu Kumar, and Tushar Krishna. "Demystifying platform requirements for diverse llm inference use cases." *arXiv preprint arXiv:2406.01698* (2024).

[3] Yan, Ran, Youhe Jiang, Wangcheng Tao, Xiaonan Nie, Bin Cui, and Binhang Yuan. "FlashFlex: Accommodating Large Language Model Training over Heterogeneous Environment." *arXiv preprint arXiv:2409.01143* (2024).

[4] Liu, Siqin, Prakash Chand Kuve, and Avinash Karanth. "HSCONN: Hardware-Software Co-Optimization of Self-Attention Neural Networks for Large Language Models." In *Proceedings of the Great Lakes Symposium on VLSI 2024*, pp. 736-741. 2024.

[5] He, Pujiang, Shan Zhou, Wenhuan Huang, Changqing Li, Duyi Wang, Bin Guo, Chen Meng, Sheng Gui, Weifei Yu, and Yi Xie. "Inference performance optimization for large language models on cpus." *arXiv preprint arXiv:2407.07304* (2024).

[6] Ditto, P. S., and V. G. Jithin. "Inference Acceleration for Large Language Models on CPUs." (2024).

[7] Li, Xinjin, Yu Ma, Yangchen Huang, Xingqi Wang, Yuzhen Lin, and Chenxi Zhang. "Synergized Data Efficiency and Compression (SEC) Optimization for Large Language Models." In *2024 4th International Conference on Electronic Information Engineering and Computer Science (EIECS)*, pp. 586-591. IEEE, 2024.

[8] Yu, Zhongkai, Shengwen Liang, Tianyun Ma, Yunke Cai, Ziyuan Nan, Di Huang, Xinkai Song et al. "Cambricon-llm: A chiplet-based hybrid architecture for on-device inference of 70b llm." In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1474-1488. IEEE, 2024.

[9] Liang, Bor-Sung. "Computing Architecture for Large-Language Models (LLMs) and Large Multimodal Models (LMMs)." In *Proceedings of the 2024 International Symposium on Physical Design*, pp. 233-234. 2024.

[10] Duan, Gaoxiang, Jiajie Chen, Yueying Zhou, Xiaoying Zheng, and Yongxin Zhu. "Large Language Model Inference Acceleration Based on Hybrid Model Branch Prediction." *Electronics* 13, no. 7 (2024): 1376.

[11] Zhang, Hengrui, August Ning, Rohan Prabhakar, and David Wentzlaff. "A hardware evaluation framework for large language model inference." *arXiv preprint arXiv:2312.03134* (2023).

[12] Isaev, Mikhail, Nic McDonald, Larry Dennison, and Richard Vuduc. "Calculon: a methodology and tool for high-level co-design of systems and large language models." In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1-14. 2023.

[13] Emani, Murali, Sam Foreman, Varuni Sastry, Zhen Xie, Siddhisanket Raskar, William Arnold, Rajeev Thakur, Venkatram Vishwanath, and Michael E. Papka. "A comprehensive performance study of large language models on novel ai accelerators." *arXiv preprint arXiv:2310.04607* (2023).

[14] Ye, Zhengmao, Dengchun Li, Jingqi Tian, Tingfeng Lan, Jie Zuo, Lei Duan, Hui Lu et al. "Aspen: High-throughput lora fine-tuning of large language models with a single gpu." *CoRR* (2023).