Identifying bottlenecks in Mobile development of LLM's

Saad Momin Mehmood
Lahore University of Management Sciences
Lahore, Punjab, Pakistan

Huzaifa Ahmad
Lahore University of Management Sciences
Lahore, Punjab, Pakistan

Muhammad Ali Hassan
Lahore University of Management Sciences
Lahore, Punjab, Pakistan

1 INTRODUCTION

Large Language Models (LLMs) have emerged as the cornerstone of modern computational devices, driving advancements in natural language processing (NLP) with their ever-growing complexity and capabilities. These State-of-the-Art models have redefined tasks such as text generation, semantic understanding, and contextual analysis, enabling unprecedented functionality on diverse platforms.

Llama 3.2, with its 1 billion parameters, exemplifies the potential of LLMs for mobile and edge computing applications. However, deploying such models on resource-constrained devices, such as smartphones or IoT hardware, presents significant challenges due to limitations in memory capacity and processing power. In our study, we operate under the stringent constraint of 2-bit quantization to accommodate the hardware capabilities of these devices.

This work investigates the memory footprint and operational bottlenecks of deploying State-of-the-Art LLMs on constrained hardware. By systematically recording metrics for memory utilization, such as active and inactive memory, cached data, and swap usage during model execution, we uncover key performance trends. The insights gained highlight critical areas for optimization, providing a roadmap for efficient memory management and resource allocation in edge environments.

Our findings contribute to the understanding of LLM feasibility in constrained devices, offering valuable perspectives on optimizing real-world applications. This report lays the groundwork for future strategies to improve the performance and scalability of LLMs in edge computing scenarios. Implementation details and additional results are available in our public repository [1].

2 PROBLEM DESCRIPTION

Deploying State-of-the-Art (SoTA) Large Language Models (LLMs) on resource-constrained devices poses a significant technical challenge. While LLMs, such as Llama 3.2 with 1 billion parameters, have revolutionized natural language processing by enabling advanced tasks like text generation and semantic understanding, their computational demands often exceed the capabilities of edge devices like smartphones and IoT hardware.

Resource limitations, including reduced memory capacity, lower processing power, and restricted energy efficiency, constrain the feasibility of running these models on mobile devices. Further, the necessity of techniques like 2-bit quantization to accommodate hardware constraints introduces additional performance trade-offs, often affecting response accuracy and latency. These challenges are compounded by a lack of detailed insights into memory behavior, such as allocation patterns, active and inactive usage, and swap memory utilization, during the operation of LLMs in constrained environments.

This study addresses these challenges by investigating the operational bottlenecks and memory footprint associated with deploying LLMs on resource-constrained devices. By systematically analyzing memory metrics, this work aims to provide actionable insights for optimizing resource allocation and ensuring the efficient execution of LLMs on edge platforms.

3 MOTIVATION

The rapid advancements in Large Language Models (LLMs) have opened up transformative possibilities for natural language processing, particularly with models like Llama 3.2, which pushes the boundaries of performance with 1 billion parameters. However, the deployment of such State-of-the-Art models on resource-constrained devices, such as smartphones and IoT platforms, remains an underexplored frontier. Despite their immense potential, the high computational demands and memory requirements of these models often render them inaccessible for edge environments.

This study is motivated by the dual need to bridge this accessibility gap and establish new benchmarks for running LLMs on constrained hardware. With the release of Llama 3.2, this research represents the first of its kind to rigorously analyze its feasibility on mobile devices under stringent hardware limitations, such as 2-bit quantization. The

findings aim to provide critical insights into memory management, latency, and computational bottlenecks that arise in real-world scenarios.

Furthermore, understanding these constraints has implications far beyond academic interest. As edge computing continues to expand in low-resource and remote areas, making advanced AI technologies like LLMs more accessible can democratize their use, enabling innovations in healthcare, education, and communication. By offering a detailed exploration of the challenges and opportunities of deploying LLMs on constrained devices, this study not only pioneers a path for future research but also sets the stage for optimizing LLM performance in edge environments.

4 METHODOLOGY

To evaluate the feasibility of deploying Large Language Models (LLMs) on resource-constrained devices, we implemented a systematic approach to measure and analyze the performance of Llama 3.2 with 1 billion parameters on an Android-based edge device. The methodology focused on understanding memory usage, identifying operational bottlenecks, and capturing detailed system metrics during model execution. The process consisted of the following stages:

4.1 Environment Setup

- **Hardware Configuration:** The study utilized a resource-constrained Android smartphone as the edge device. The specifications of the device are in Table 1.
- **Software Environment:** Termux with Proot-Distro Debian was employed to create a Linux-like environment on the device. The Llama 3.2 model was executed using the Ollama tool.
- **Connectivity:** The Android device was connected to a PC via a USB cable, enabling communication through SSH. This setup facilitated real-time data collection and monitoring using Android Debug Bridge (ADB).

Specification	Details
Model	LG Nexus 5X
Model Numbers	LG-H790 (North America), LG-H791 (International)
Launch Year	2015
RAM	2 GB
Android Version	Initially launched with Android 6.0 Marshmallow, upgradable to Android 8.1 Oreo
Storage/ROM	32 GB

Table 1: Specifications of the Mobile Device Used for LLM Deployment

4.2 Execution Pipeline and Automation

To ensure a controlled and reproducible experimental workflow, we established a semi-automated execution pipeline that integrated remote monitoring, data logging, and prompt evaluation. The pipeline leveraged a combination of Android Debug Bridge (ADB) and Secure Shell (SSH) communication channels, facilitated by Termux with Proot-Distro Debian, to execute the LLM and record performance metrics in real time. The key steps were as follows:

- **Remote Access Configuration:** The edge device (Android smartphone) was connected to a host PC via a USB cable. SSH access was enabled through Termux, allowing for seamless command execution on the mobile device. This setup enabled simultaneous operations: the host PC could dispatch control signals, while the mobile device ran the Ollama-based LLM instance.
- **Memory Utilization Monitoring:** A dedicated monitoring script on the host PC, triggered via SSH commands, utilized ADB to capture system memory metrics from the Android device at predefined intervals. This included total memory, active and inactive memory pages, cached data, and swap usage.
- **Automated Start/Stop Protocol:** Execution of the LLM model and data collection was orchestrated through a series of SSH signals. Before launching the LLM, a baseline memory utilization was recorded for 5 seconds to establish a reference. Subsequently, an SSH "start" signal initiated the memory logging on the PC. The LLM was then executed on the Android device, and response latencies were measured in parallel. Throughout the LLM's operation, memory metrics were continuously monitored and recorded.
- **Terminating and Finalizing Measurements:** Once sufficient data had been collected—covering both runtime performance and latency—a termination signal was sent via SSH to halt the LLM execution. Memory usage continued to be monitored for an additional 5 seconds post-execution to assess lingering effects. This allowed for a comprehensive evaluation of the model's lifecycle performance, from pre-run baseline conditions through execution and after termination.
- **Scripting and Automation:** To minimize manual overhead and improve reproducibility, the workflow was encapsulated in Bash scripts deployed on the mobile device. Separate scripts were developed for each prompt category, streamlining the initiation of the LLM and the capture of memory metrics. This modular approach facilitated rapid iteration and reconfiguration of experimental scenarios.

By integrating these automated procedures into the experimental design, we ensured consistent data collection, streamlined execution, and facilitated the systematic analysis of LLM performance under constrained mobile environments.

4.3 Data Processing and Analysis

The raw memory data was processed to identify patterns and bottlenecks in system performance:

- The metrics were parsed and organized in CSV format to enable efficient analysis and visualization.
- Memory trends were plotted to highlight variations in active, inactive, and cached memory during different phases of model operation.
- Comparative analyzes were conducted to correlate memory usage with observed latency and system stability.

4.4 Evaluation of Bottlenecks

The collected data was used to pinpoint performance constraints, such as excessive swap usage or uneven memory allocation, that limit the viability of deploying LLMs on constrained hardware. The insights gained from these evaluations offer actionable recommendations for optimizing memory management and model execution strategies.

Prompt Category	Example Prompt
Explanation/Definition	Explain the water cycle in simple terms.
Essay Writing	The model was asked to write a 200-300 worded essay.
Summary	A passage was given, and the model was asked to summarize it.

Table 2: Categories of Prompts

5 RESULTS AND DISCUSSIONS

Table 2 presents the three distinct prompt categories utilized in our evaluation. Each prompt type was associated with a different context length, thus enabling a comparative assessment of the model's performance under varying input complexities. By systematically varying the prompt complexity and length, we aimed to uncover the model's ability to maintain coherence, handle longer dependencies, and sustain its performance as the contextual scope increased.

5.1 Prompt categories

The chosen prompts in Table 2 represent a progression in context length and complexity, thus providing a structured basis to assess LLM performance on a resource-constrained mobile device. Each prompt was specifically selected to stress different aspects of the model's reasoning, memory utilization, and linguistic capabilities under increasing input demands.

- **Prompt 1 (Short Context):** *"Explain the water cycle in simple terms."* This prompt is succinct, consisting of a single, straightforward query that requires the model to recall basic factual information and present it clearly. The short context length ensured minimal computational overhead, providing a baseline measurement of the model's latency and memory footprint when handling a simple, direct request. On the constrained device, the LLM produced a concise explanation most of the time but also hallucinated other times. There was variability in performance aswell with some prompts seeing performance degradation while others had minimal degradation with each prompt taking roughly 1-2 minutes to run.
- **Prompt 2 (Moderate Context):** *"I need you to write me an essay on why slaves were important to the economy in the past..."* This prompt introduces a higher degree of complexity, featuring specific structural requirements (200–300 words, multiple paragraphs) and a nuanced historical-economic topic. The longer input and detailed instructions test the model's capacity to maintain coherence, follow multi-step directives, and manage intermediate reasoning over an extended response. On the android device, the LLM experienced a modest increase in latency and memory utilization, and it began to hallucinate a lot aswell. There was performance degradation with more outliers and variability being seen in the LLM response with each prompt taking roughly 5-6 minutes to run.
- **Prompt 3 (Long Context):** *"Analyze and summarize the provided narrative about Echo Prime..."* Here, the model is presented with a lengthy narrative describing an interstellar expedition, detailed character backgrounds, and a complex, sentient environment. The prompt requests a structured summary, requiring the model to process and condense a large body of text while preserving critical narrative elements and thematic coherence. This extensive context length significantly stressed the system's memory and processing capacity. On the edge device, the LLM took noticeably much longer to generate a response and required more memory resources, and had mostly hallucinated after the first few characters generated. Performance overhead was more pronounced compared to shorter prompts, and it took over 25 minutes in general for each prompt to run.

5.2 Analyzing the Graphs

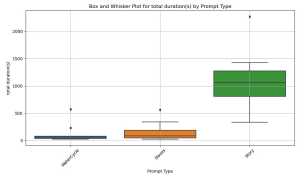
Figures 1 present six box-and-whisker plots that capture performance metrics across three prompt categories: a simple factual query (Prompt 1), a moderately structured essay (Prompt 2), and a lengthy narrative (Prompt 3). As seen in Figure (a), total duration scales with increasing prompt complexity, starting low for Prompt 1 and rising substantially for Prompts 2 and 3, indicating heavier computational loads and memory usage for more elaborate inputs.

In contrast, the interpretation of load duration (Figure (b)) must account for differing measurement conditions. For the simplest prompt (WaterCycle), Ollama's prompt-level caching ensured that the model was consistently loaded in memory, resulting in minimal and stable load durations. However, for Prompt 2 (Slaves), breaks taken between measurements disrupted caching efficiency, leading to erratic and elevated load times. In Prompt 3 (Story), Termux crashes forced the model to reload frequently, preventing any caching benefit and resulting in persistently high but somewhat more controlled load durations than Prompt 2's fragmented scenario.

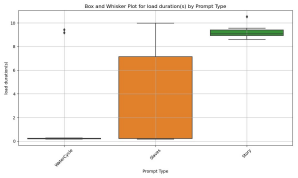
Figure (c) shows evaluation counts increasing and becoming more variable as prompt complexity grows, reflecting the mounting difficulty of maintaining coherent, stable processing over extensive context windows. Similarly, Figure (d) indicates that evaluation duration per step not only increases but also becomes more unpredictable with complexity, suggesting heavier internal overheads under constrained resources.

In summary, while prompt complexity inherently drives up resource demands and processing overheads, the observed variability in load and evaluation rates is also tied to caching mechanisms and testing conditions. Ensuring consistent caching and streamlining the input process can reduce load irregularities, while previously discussed optimizations—memory management, incremental processing, model compression, and specialized inference runtimes—remain essential for maintaining stable and efficient performance on resource-constrained mobile hardware.

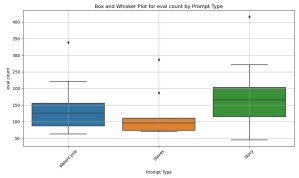
Analysis of the line graphs: Figure 2a plots evaluation rate against the number of tokens in the prompt. At lower token counts, the evaluation rate varies substantially, swinging between approximately 1.6 to nearly 3.0 tokens/s, reflecting a more erratic and potentially resource-dependent response. As the prompt size grows and surpasses several hundred tokens, the evaluation rate settles into a narrower band, dropping in overall speed but becoming more predictable. This stabilization suggests that once the model reaches a certain scale of input, the overheads and memory pressures become relatively constant, producing more uniform but consistently lower performance. In other words, smaller prompts lead to faster but less stable throughput, whereas large prompts yield a slower yet more consistent evaluation rate.



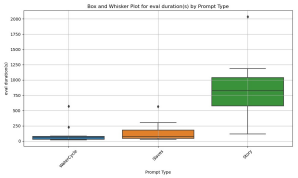
(a) Total Duration by Prompt Type



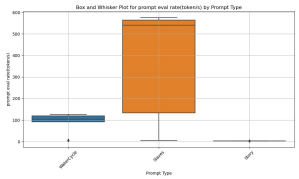
(b) Load Duration by Prompt Type



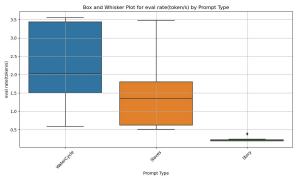
(c) Evaluation Counts by Prompt Type



(d) Evaluation Duration by Prompt Type



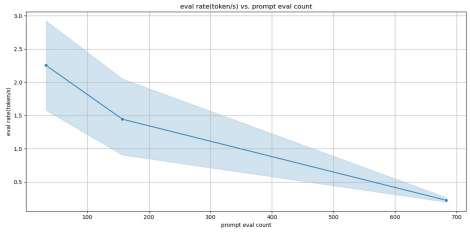
(e) Prompt Evaluation Rate by Prompt Type



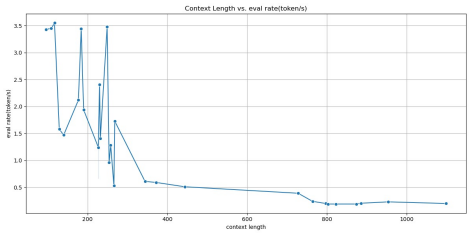
(f) Evaluation Rate by Prompt Type

Figure 1: Box and Whisker Plots of Various Performance Metrics by Prompt Type. Each plot compares distributions of metrics (e.g., evaluation count, load duration) across the three prompt categories detailed in Table 2.

Figure 2b, showing evaluation rate as a function of context length, reveals a similar trend. While early intervals of lower context lengths exhibit sharp fluctuations, beyond a certain threshold, the evaluation rate plateaus into a narrow, steady range. Here again, the model transitions from a variable regime—where differences in hardware or memory interactions can cause significant rate swings—to one where complexity-induced constraints have fully taken hold, resulting in a stable but reduced evaluation rate. These patterns indicate that while simpler tasks can benefit from occasional bursts of higher throughput, scaling up input size eventually imposes steady, uniformly distributed strain on the system's limited resources, leading to more predictable but consistently lower performance.



(a) Eval Rate vs. Prompt Evaluation Count



(b) Eval Rate vs. Context Length

Figure 2: Line Graphs Showing Evaluation Rate Trends as a Function of Prompt Evaluation Count and Context Length.

6 LIMITATIONS

Despite the promising insights obtained through this study, several limitations must be acknowledged:

(1) **General-Purpose Deployment on Emulated Debian:** The experimental setup relies on running an emulated Debian environment on a mobile device using Termux and Proot-Distro. While this approach facilitates experimentation, it does not cater to the unique hardware and software constraints of mobile platforms. Consequently, the findings may not directly translate to real-world, optimized mobile deployments.
(2) **Unknown Overheads of Ollama and Emulated Linux:** The use of the Ollama tool in conjunction with an emulated Linux environment introduces additional, unquantified overheads. These overheads may impact the performance measurements, and further work is required to isolate and quantify their contributions.
(3) **Context Switching Overheads from ADB:** The reliance on ADB (Android Debug Bridge) for collecting memory metrics adds context-switching overhead to the measurements. This is particularly problematic for latency measurements, as higher context lengths amplify the effects of these overheads, making it challenging to accurately gauge true latency performance.
(4) **Limited Mobile Functionality and Specialized Deployment Challenges:** The current mobile deployment setup is highly constrained in terms of functionality. Implementing a specialized and efficient deployment strategy that fully leverages the hardware capabilities of mobile devices would require significant additional effort and customization, which lies beyond the scope of this study.

Addressing these limitations will be critical in future work to enhance the applicability and accuracy of LLM deployment on resource-constrained mobile devices.

7 CONCLUSION AND FUTURE WORK

Building upon our findings, the integration of Meta's Llama Stack framework presents a compelling avenue for addressing the bottlenecks identified in deploying large language models (LLMs) on resource-constrained mobile devices. Llama Stack offers a modular, standardized approach to generative AI development, which aligns well with our objectives of optimizing memory utilization and computational efficiency under constrained environments.

Additionally, we have collected data on memory footprints, including detailed memory usage statistics. However, due to time constraints, we were unable to analyze this data comprehensively or visualize it through graphs. Future work will focus on processing and interpreting this dataset to provide deeper insights into memory behavior and optimization strategies.

7.1 Optimizing Memory and Inference

The Memory API in Llama Stack could significantly enhance the way memory is managed during inference, enabling efficient storage and retrieval of contextual data. This would address the excessive swap usage and memory fragmentation observed in our experiments, particularly under high-complexity prompts. Moreover, the Inference API's support for batch requests and configurable model behavior could optimize token processing rates, reducing latency while maintaining coherence.

7.2 Enhancing Safety and Contextual Integrity

Deploying the Safety API could mitigate hallucinations and inaccuracies by introducing robust content moderation layers. Additionally, its configurability for different violation levels could be tailored to the sensitive applications of LLMs in healthcare or education, expanding the usability of LLMs in real-world scenarios.

7.3 Agent Integration for Functional Expansion

By leveraging the Agent API, our deployment strategy could incorporate external tools and memory banks, empowering LLMs to dynamically access external databases or perform contextually aware computations. This would enhance the functional breadth of LLMs without increasing their direct computational burden.

7.4 Future Deployment Strategies

Llama Stack's turnkey deployment solutions could streamline the process of transitioning our experimental setup to optimized production environments, such as on-premises servers or public cloud platforms. This would ensure that the benefits of our study translate seamlessly into scalable, real-world applications.

7.5 Potential for Standardization and Collaboration

Adopting Llama Stack fosters a service-oriented architecture that supports interoperability and composability. This aligns with our vision of standardizing LLM deployment practices for edge computing, facilitating broader collaboration across research and industrial applications.

8 RELATED WORKS

The deployment of large language models (LLMs) on mobile devices has garnered increasing attention due to privacy concerns and the desire for offline functionality in various applications. Recent research efforts have explored different strategies to overcome the hardware limitations of mobile devices, focusing on performance optimization, model quantization, and efficient processing frameworks.

Performance Benchmarking: A thorough evaluation of LLM performance on mobile platforms was conducted by Xiao, Huang, Chen, and Tian, as documented in their study titled "Large Language Model Performance Benchmarking on Mobile Platforms: A Thorough Evaluation" [7]. Their research provides a comprehensive analysis of lightweight LLMs, such as Gemini Nano and LLAMA2 7B, on commercial-off-the-shelf mobile devices. They assess critical metrics like token throughput, latency, and battery consumption, as well as resource utilization and the impact of hardware capabilities on model performance. This benchmarking offers valuable insights for developers aiming to optimize LLMs for mobile deployment[2].

Model Quantization: Tan, Lee, Dudziak, and Xu's work in "MobileQuant: Mobile-friendly Quantization for On-device Language Models" [5] introduces MobileQuant, a post-training quantization method tailored for mobile devices. This approach addresses the challenge of efficiently deploying LLMs on mobile hardware by significantly reducing both inference latency and energy consumption, all while preserving model accuracy comparable to those achieved with 16-bit activations. This approach leverages the capabilities of existing mobile hardware, particularly NPUs, making it an attractive solution for on-device deployment.

Processing Frameworks: The development of efficient processing frameworks for LLMs on mobile devices is explored in the paper "Efficient Inference on Mobile Devices for Large Language Models" by Wang, Zhang, and Liu [6]. They propose a novel framework, "EdgeLLM", which optimizes memory usage and computational efficiency through techniques like model pruning and dynamic loading. EdgeLLM allows for real-time processing capabilities on low-power mobile devices, demonstrating significant improvements in processing speed and memory efficiency.

User Experience and Usability: Focusing on the user experience, the study by Patel and Kumar, "User Experience in Mobile AI Applications: A Case Study with Large Language Models" [4] (2024), delves into how LLMs can enhance the interaction quality in mobile applications. Their research highlights that the integration of LLMs not only aids in providing more personalized responses but also significantly improves usability by reducing user frustration through better context awareness and response accuracy. This case study not only underscores the potential of LLMs to transform user interactions by making them more natural and intuitive but also outlines some challenges such as the need for robust privacy measures to protect user data, given the depth of interaction LLMs facilitate.

PALMBench: PALMBENCH (Li et al., 2024) [3] introduces a comprehensive automated framework for evaluating compressed LLMs on mobile platforms. It benchmarks resource efficiency, latency, and energy consumption while analyzing the impact of quantization on performance and toxicity. This study complements existing works by addressing unexplored challenges, such as inference efficiency and memory constraints, aligning closely with our objective of deploying LLMs in resource-constrained environments. It offers valuable insights for optimizing models for mobile devices and extends the conversation on hardware-aware LLM evaluation.

REFERENCES

[1] 2024. CS6303 Project: LLM feasibility in edge devices - Implementation & Results. https://github.com/huzaifa-abajwa/CS6303-Project. (2024).
[2] Hannes Fassold. 2019. Porting Large Language Models to Mobile Devices for Question Answering. Steyrergasse 17, 8010 Graz. Tel.: +43 316 876-1126, e-mail: hannes.fassold@joanneum.at.
[3] Yilong Li, Jingyu Liu, Hao Zhang, M Badri Narayanan, Utkarsh Sharma, Shuai Zhang, Pan Hu, Yijing Zeng, Jayaram Raghuram, and Suman

Banerjee. 2024. PALMBENCH: A Comprehensive Benchmark of Compressed Large Language Models on Mobile Platforms. arXiv preprint arXiv:2410.05315. https://doi.org/10.48550/arXiv.2410.05315

[4] Patel and Kumar. 2024. User Experience in Mobile AI Applications: A Case Study with Large Language Models. *Journal of Mobile AI* Volume Number, Issue Number, First Page–Last Page.

[5] Fuwen Tan, Royson Lee, Łukasz Dudziak, and Shell Xu Hu. 2024. MobileQuant: Mobile-friendly Quantization for On-device Language Models. arXiv preprint arXiv:2408.13933. arXiv. https://doi.org/10.48550/arXiv.2408.13933

[6] Wang, Zhang, and Liu. 2024. Efficient Inference on Mobile Devices for Large Language Models. Unpublished manuscript.

[7] Jie Xiao, Qianyi Huang, Xu Chen, and Chen Tian. 2024. Large Language Model Performance Benchmarking on Mobile Platforms: A Thorough Evaluation.