Poster: Distributed LLMs: Partitioning LLMs for Distributed On-Device Reasoning

Reo Kuchida*
University of Tartu
Tartu, Estonia
reo.kuchida@ut.ee

Huber Flores
University of Tartu
Tartu, Estonia
huber.flores@ut.ee

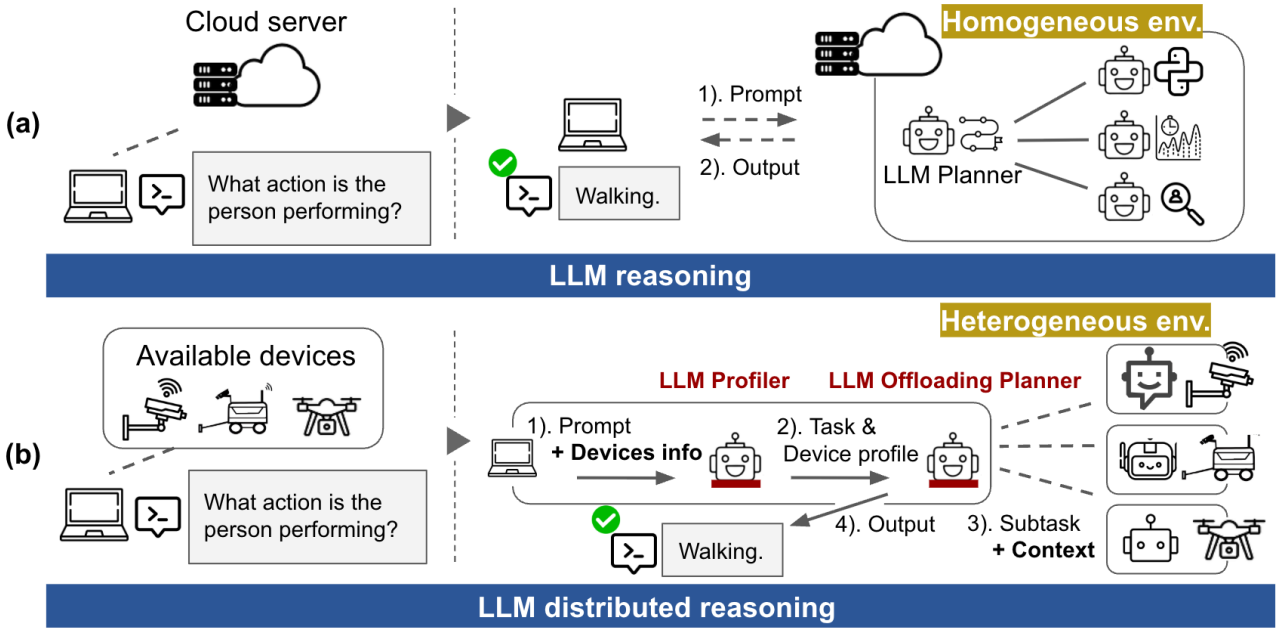


Figure 1: (a) Multi-agent LLM reasoning in homogeneous env. (b) LLM distributed reasoning in heterogeneous env.

1 INTRODUCTION

On-device LLMs are becoming feasible, but prolonged inference times and high energy consumption on resource-constrained devices motivate collaborative distributed LLM reasoning through offloading. The challenge is that LLM inference exhibits tightly coupled computation, where each generated token depends on previous tokens and the evolving context, including conversation history. This dependency does not fit traditional computation offloading approaches based on code execution characteristics. Existing multi-agent frameworks decompose reasoning workloads across multiple agents, however, they typically assume homogeneous environments where agents share a common memory space or operate under a single global context [1]. To address these challenges, LLMOffload [2] is introduced, demonstrating distributed LLM reasoning across heterogeneous environments by embedding relevant context into offloaded subtasks. LLMOffload reduces encoding latency by approximately 85% compared with naively transferring the full conversation history while preserving task accuracy through efficient context embedding. Our work paves the way toward distributing on-device LLM reasoning across heterogeneous distributed environments.

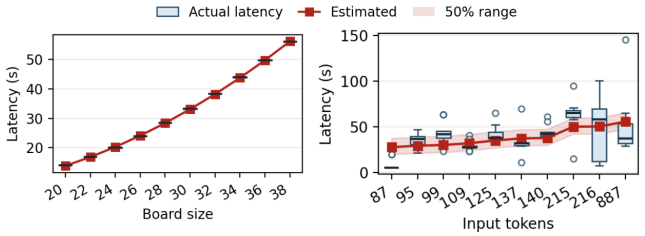
2 UNIQUENESS OF LLM WORKLOAD

The context-dependent nature of LLM workloads makes offloading across different environments difficult, motivating new approaches such as LLMOffload. We first highlight the difference between a conventional computational workload (minimax algorithm) and an LLM reasoning workload (MMLU-Pro).

Coupled reasoning process: The conventional computational workload can be divided using static code analysis because each recursive branch takes an independent state as input and returns its output to the parent. In contrast, Transformer-based autoregressive generation depends on the input prompt and previously generated tokens. This sequential dependency tightly couples the reasoning process and limits its decomposability.

Unpredictable complexity: Figure 2 shows the relationship between input and latency on Raspberry PI 4. Following the MAUI cost estimation approach, we use a polynomial regression model to estimate task latency. The mean relative error is about 0.48% for the minimax workload, but 79% for the LLM reasoning workload. This shows that latency prediction is challenging for LLM reasoning because its workload complexity is sensitive to prompt variations and difficult to predict before execution.

*Corresponding Author



(a) MInimax algorithm workload (b) LLM reasoning workload

Figure 2: Latency distribution: conventional vs. LLM.

3 DISTRIBUTED LLM REASONING

LLMOffload enables LLM workload offloading by defining LLM subtasks. Each subtask is formulated as a prompt-based unit enriched with the context of the conversation, and is offloaded to available heterogeneous devices. LLMOffload uses two LLM components to construct context-embedded subtasks and determine their offloading: an *LLM profiler* and an *LLM offloading planner*.

LLM profiler & LLM offloading planner: To decide where to offload subtasks, LLMOffload require information about the computational performance of each device, as well as the model performance of the local LLM deployed on that device, to ensure the quality of the outputs produced by offloaded subtasks. The LLM profiler first computes *device-capacity* and *model-capacity* scores, each ranging from 0.0 to 1.0, by combining device metadata, such as the CPU name and model name, with background knowledge. Moreover, the LLM profiler generates a task profile that helps the LLM offloading planner construct subtasks by characterizing task-specific properties such as decomposability, cross-boundary coupling, and reasoning horizon. Using the profiles, the LLM offloading planner generates context-embedded, device-specific subtasks and assigns them to efficient devices based on computation and model capacity.

4 THE EXPERIMENT

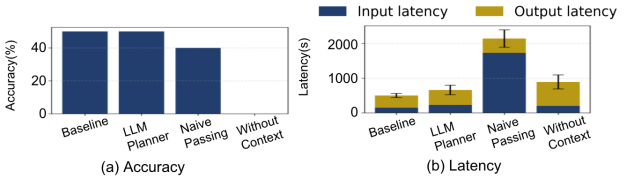
We use video-based question answering workloads from MVBench and create follow-up QA tasks by rewriting questions across task types for the same video. Object mentions are replaced with pronouns (e.g., "it"), making prior answers necessary to resolve ambiguity. This evaluates whether the planner preserves context across subtasks. Table 1 provides examples.

Apparatus: We deploy four heterogeneous worker devices: Raspberry Pi 4 Model B, HP EliteBook 840 G6, GPU-accelerated Jetson Orin Nano, and M1 MacBook Air. All workers run the same local LLM, Qwen3.5-4B, via Ollama to isolate the impact of context embedding. Devices connect to a Wi-Fi AP and report their device and model names using `lscpu` and `ollama list`, respectively.

Procedure: The coordinator stores the MVBench dataset and manages follow-up QA tasks. It first discovers the four worker devices on the same network, then uses LLMOffload to offload subtasks across them and aggregate their outputs into a final answer. We compare three distributed execution settings: (i) *LLM planner*, which embeds the necessary context into each follow-up subtask; (ii) *naive context passing*, where the full conversation history is appended to the follow-up question and sent to each worker; and (iii) *without context*, where only the ambiguous follow-up question is sent to the workers. We compare these settings against a *baseline* that uses the original non-ambiguous question.

Table 1: Examples of questions from our workload.

Question	Candidate	Answer
`task type: object interaction (original)`		
Which object was sat at by the person?	The bed. / The sofa. The couch. / The table.	The table.
`task type: action prediction (follow-up)`		
What will the person do next with **it (the table)**?	Take. / Tidy up. Sit at. / Wash.	Sit at.



(a) Accuracy (b) Latency

Why did the person take the object? Use ONLY the frames in this segment.
Context: We're analyzing this video to find the reason for object taking. In the previous step, we confirmed the object was 'the phone/camera'. Now, briefly explain the reason using only this segment's frames.
Return ONLY JSON:
{"choice": "<LETTER or UNKNOWN>", "confidence": 0.0 to 1.0, "evidence": "<1 sentence>", "notes": "<optional>"}" }

(c) Example of subprompt in the follow-up question

Figure 3: Performance and latency on follow-up question workload across methods.

Results: Figure 3 shows that context embedding enables workers to resolve anaphoric references in follow-up questions. The LLM planner preserves the baseline accuracy of 50% by embedding only the necessary context, such as the resolved object from the previous step, into each subprompt. Without context, accuracy drops to 0%, while naive context passing recovers 40% accuracy but incurs much higher latency (8×) by encoding the full conversation history.

Roadmap ahead: A current limitation is suboptimal task assignment due to inaccurate device profiles. The profiler assigns capacity scores of 0.85, 0.75, 1.0, and 0.2 to the MacBook, Jetson Orin Nano, EliteBook, and Raspberry Pi, respectively, excluding the Raspberry Pi from offloading. As a result, the Jetson Orin Nano finishes in 502.9s, while the EliteBook becomes the bottleneck at 3207s. Benchmark-driven profiling and historical execution data could improve capacity estimation and workload allocation. Beyond this, distributed on-device LLMs introduce new challenges, including: 1) Unified scheduling, integrating conventional computation and LLM workloads within a common offloading framework; 2) Edge–cloud distributed reasoning, enabling dynamic migration of LLM workloads across the edge–cloud continuum; and 3) opportunistic distributed inference, exploiting dynamic edge resources through adaptive workload placement and migration.

ACKNOWLEDGMENTS

This work is funded by Estonian Research Council grant (PRG 2725).

REFERENCES

[1] Taicheng Guo et al. 2024. Large language model based multi-agents: a survey of progress and challenges. In *Proceedings of the 33rd ACM IJCAI 2024.*
[2] Reo Kuchida et al. 2026. LLMOffload: Cooperative Offloading for Efficient On-device LLM Reasoning. In *Proceedings of the 23rd ACM EWSN 2026.*