LLMOffload: Cooperative Offloading for Efficient On-Device LLM Reasoning

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

Zhigang Yin
University of Tartu
Tartu, Estonia
zhigang.yin@ut.ee

Abdul-Rasheed Ottun
University of Tartu
Tartu, Estonia
rasheed.ottun@ut.ee

Kevin Post
University of Tartu
Tartu, Estonia
kevin.post@ut.ee

Kostiantyn Voskovtsov
University of Tartu
Tartu, Estonia
kostiantyn.voskovtsov@ut.ee

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

ABSTRACT

LLMs are rapidly transitioning from cloud-based deployments to on-device execution, enabling intelligent decision making closer to data sources while exposing new challenges for resource-constrained devices. Effective LLM inference often relies on prolonged reasoning processes that incur high computational and energy costs, making sustained on-device execution impractical. While computational offloading offers a natural mechanism to mitigate these limitations, on-device LLM workloads exhibit unique characteristics related to prompting structure, token usage, iterative reasoning, and context sharing that challenge existing offloading strategies. We propose LLMOffload, an on-device LLM framework for cooperative offloading that decomposes LLM task reasoning into asynchronous subtasks and distributes them across heterogeneous devices using structured prompts that preserve intermediate reasoning state and end-to-end logical coherence. To decide when and how to offload, LLMOffload employs an LLM-based profiler and planner that reason over task prompts and worker context to synthesize subtask and aggregation prompts, even under incomplete or previously unseen device and model metadata. Through rigorous evaluation against on-device and cloud-based baselines, we demonstrate that LLMOffload reduces on-device energy consumption by up to 65% and end-to-end latency by up to 50% compared to single device execution, while maintaining output quality. Our results highlight a practical path toward cooperative LLM reasoning, enabling prolonged LLM reasoning across heterogeneous IoT environments based on available resources and contextual constraints.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing systems and tools**.

KEYWORDS

Cloudlet; Chain-of-Thought; Tree-of-Thought: Edge AI; Agentic AI

1 INTRODUCTION

The accelerated adoption of Large Language Models (LLMs) has brought human-like reasoning capabilities into everyday applications, driving their transition from centralized cloud infrastructures to Internet of Things (IoT) and edge platforms as on-device LLMs. This evolution enables the long-standing vision of a dynamic and cognitive cloud–edge–IoT continuum, where intelligent, real-time decisions orchestrate workload execution across heterogeneous and resource-constrained hardware, unlocking advanced applications such as real-time surveillance [44], traffic control [39], and vision-based systems for service robots and autonomous drones [47]. However, despite recent advances, on-device LLM inference remains a major challenge: high-accuracy reasoning often relies on long Chains-of-Thought (CoT), which are computationally expensive and difficult to sustain under strict energy and memory constraints [6, 23, 31]. As a result, devices may become trapped in prolonged inference processes, rapidly draining battery life and limiting long-term deployment. This recurring issue highlights the need for cooperative and adaptive inference on-device strategies that reduce reasoning overhead while preserving decision quality.

Existing cooperative approaches primarily rely on computational offloading to distribute processing complexity from constrained devices to more capable computing infrastructure. This paradigm has been widely studied in the literature and can be realized at multiple levels [7, 13], including code, thread, and service offloading. As shown in Figure 1(a), the most common strategy delegates computation to remote cloud servers [14, 33, 43]; however, this approach critically depends on stable, high-bandwidth and low latency connectivity, which can render offloading counterproductive in dynamic or constrained environments. Alternative solutions leverage closer edge servers (cloudlets) to reduce latency, but such infrastructure remains scarce and unevenly deployed [30, 34, 61]. As shown in Figure 1(b), cooperative schemes among devices with shared spatial or temporal contexts further extend this principle by partitioning computation across multiple participants, enabling the collective execution of complex tasks. Multi-agent methods have also been explored for autonomous cooperation in task execution [45]. Agentic AI extends this paradigm by leveraging LLMs as reasoning engines, enabling agents to dynamically plan, decompose tasks, invoke tools, and adapt their execution strategy to achieve high-level goals with minimal human intervention. Distributed machine learning has exploited these cooperative mechanisms, especially for inference workloads ranging from classical neural-network [48] to Transformer-based language models [57]. Nevertheless, while computational offloading has been extensively explored for general workloads, on-device LLM reasoning introduces unique characteristics related to prompting (Figure 1(c)), such

*Corresponding Author

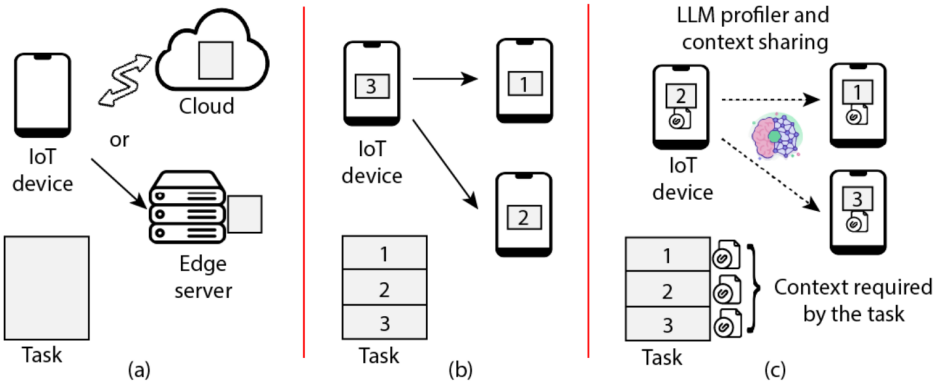


Figure 1: Paradigm shift for device offloading: (a) Device offloads to Cloud/Edge; (b) Device offloads to other nearby devices; (c) Device offloads a (LLM) task requiring shared context windows to other devices, preserving continuity and inference coherence.

as iterative inference, context window sharing, and dynamic reasoning depth, which challenge existing paradigms and motivate the exploration of new cooperative strategies tailored to distributing LLM reasoning across devices.

We introduce LLMOffload, an on-device LLM framework that cooperatively offloads reasoning complexity across multiple heterogeneous devices. LLMOffload proposes a novel reasoning pipeline that decomposes complex LLM inference into splittable, asynchronous subtasks expressed through prompts. These subtasks are enriched with semantic and device-specific heterogenous requirements, and are dynamically scheduled across surrounding devices while considering latency, energy, and privacy constraints. To enable seamless continuation of reasoning, LLMOffload captures the intermediate reasoning state, referred to as the context window or simply the context. This context includes goals, constraints, and partial results, which are packaged into the structured prompts and transferred across devices while preserving end-to-end logical coherence. Rather than partitioning token level inference within a single model, LLMOffload distributes planned reasoning steps and their intermediate artifacts, allowing devices to collaboratively advance the reasoning process. This design significantly reduces computational and energy pressure on the originating device while opportunistically leveraging more capable hardware when available. Through a rigorous comprehensive study on edge video analytics tasks, we evaluate LLMOffload using a testbed of heterogeneous devices running on-device LLMs, measuring response time and energy consumption. Our results demonstrate that cooperative execution of reasoning subtasks improves efficiency without sacrificing task-level accuracy, pointing to a promising new direction for collaborative LLM reasoning in resource-constrained IoT environments.

Summary of Contributions:

- **Novel method.** LLMOffload overcomes the unique challenges of on-device LLM tasks, enabling their offloading to heterogeneous environments without sacrificing consistency or logical inference coherence.
- **Improved performance.** LLMOffload reduces energy consumption by 65% and latency by 50% while preserving output quality.
- **Novel insights.** We demonstrate the feasibility of offloading on-device LLM tasks in heterogeneous runtime environments and show that energy savings mainly come from reduced coordination active execution time, revealing a classical offloading trade-off where benefits emerge only when computation outweighs communication and coordination overhead. We also quantify the contribution of GPU acceleration to these performance gains.

2 MOTIVATION

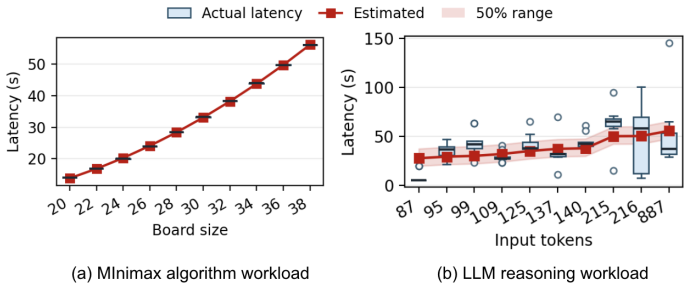
Task offloading has been widely studied across different granularities, such as code, class, and thread levels [13]. While existing profiling methods effectively evaluate the benefits of offloading to cloud or edge environments, LLM reasoning tasks exhibit distinct characteristics that challenge these approaches. As a result, they require new considerations and tailored requirements for effective offloading. We next highlight how LLM reasoning tasks differ from conventional computational workloads.

2.1 Experiment setup

Apparatus: Our testbed consists of a Raspberry Pi 4 Model B as the master device and a laptop (HP EliteBook 840 G6, 8th Gen Intel Core i5, 16GB) as the worker device. The devices are connected through a local LAN, with an average round-trip time of 102.5 ms.

Tasks: We implement two types of offloaded tasks: a conventional computation task and an LLM reasoning task. The conventional task uses a deterministic minimax algorithm that explores possible states in a board-game and returns the optimal move, representing workloads that require intensive computation while transmitting minimal data. The LLM reasoning task uses a Transformers-based inference pipeline with Qwen3-0.6B, where natural-language questions are randomly sampled from the MMLU dataset [17], and the model generates a reasoning-based response for each input.

Metrics: Each task type is evaluated over 10 trials, with execution latency measured for each run. In addition, we estimate worker-side task latency using a polynomial regression model trained on half of the collected measurements, following the profiling-based cost estimation approach of MAUI [5]. We further analyze static code complexity [37] and task decomposability [11].



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

Figure 2: Latency distribution: conventional vs. LLM.

2.2 Results

Conventional vs LLM tasks: Figure 2 compares the measured and estimated latency. The mean relative error is about 0.48% for the minimax workload, but 79% for the LLM reasoning workload. Static code analysis shows that minimax has a cyclomatic complexity of 55, and its computation grows with the square of the board size. LLM inference has a higher cyclomatic complexity of 88, its computation cost is mainly driven by the generated output size, which shows high variability across runs: min/mean/max = 18/117/223 tokens, with a coefficient of variation of 47.6%. By looking at data-dependency, minimax computational task is decomposable because each recursive branch takes an independent board state as input and returns a score to the parent. In contrast, LLM reasoning is sequentially dependent because each generation step takes the previously generated context as input. As a result, LLM reasoning tasks are non-separable, globally coupled inference workloads.

Key differences: Our results show that workload and latency prediction is challenging due to sensitivity to prompt variations and strong token dependencies. In LLM reasoning, this is amplified by autoregressive generation, where each token depends on the input prompt and previously generated tokens. This sequential dependence limits decomposability, as inference cannot be split into independent units and instead forms a globally coupled computation. Consequently, both execution load and latency are difficult to predict due to variability in generation length and stochastic sampling during decoding. This thus motivates the need for approaches for LLM decomposition and offloading.

3 METHODOLOGY

Since LLM reasoning tasks are non-separable and differ from conventional deterministic workloads, effective offloading requires decomposing them into subtasks using context-capturing mechanisms that preserve inference continuity. This allows inference to be paused, transferred, and resumed across devices without breaking the reasoning flow. LLMOffload defines LLM subtasks (or simply subtasks) as prompt-based units enriched with semantic structure and dynamic profiling factors (e.g., latency, CPU), enabling efficient matching of each subtask to the most suitable device for offloading.

Technical challenges: We address two key challenges when decomposing an LLM reasoning task into subtasks for execution by workers. First, multi-step reasoning requires preserving logical continuity, meaning task decomposition must be carefully selected so that aggregating subtasks produces outputs equivalent to monolithic execution; otherwise, coherence is lost. Second, subtasks must be accompanied by selective context window sharing that includes only information relevant to the assigned subtask. This enables a coordinator to offload computation without requiring workers to maintain the full reasoning history. Efficiently structured prompts (see Table 1) that carry only essential context with minimal payload are therefore critical to reducing coordination overhead. Naive partitioning of complex workloads can disrupt shared context, preventing workers from building on prior reasoning and ultimately degrading answer quality.

LLMOffload overview: LLMOffload employs a coordinator-worker architecture in which a coordinating node orchestrates the offloading operation. As illustrated in Figure 3(a), LLMOffload initiates when a user submits a prompt. As shown in Figure 3(b), the prompt is executed through a three-phase operational pipeline consisting of: (i) LLM profiler, (ii) LLM offloading planner, and (iii) scheduling and execution. At the initial phase, the coordinator node initiates the *LLM profiler* that characterizes the profiles of the task and available devices (workers) to generate structured profiles regarding (a) the task feature vector and (b) the conditions and estimated capacities of available workers. This can extract task dependency and decomposability descriptors from the task prompt, and normalize heterogeneous worker capacities into comparable scores. In the next phase, the *LLM offloading planner* decides the offloading strategy based on the decomposition rule. It selects a task-appropriate strategy (e.g., temporal vs. spatial segments) and generate device specific subtask prompts. Each subtask prompt embeds work-assignment context to preserve the original task intent and constraints, helping to maintain answer quality under distributed condition. An aggregation prompt is also generated for final response. In the final phase, the coordinator runs a lightweight scheduling algorithm that leverages the *LLM profiler*'s worker capacity estimates to compute an optimal task-to-worker allocation following to the *LLM offloading planner*'s strategy. It then dispatches the subtasks, collects the outputs, combine the returned results into a single response by applying the aggregation prompt, which is sent back to the user.

Phase I-LLM profiler: Decomposing complex reasoning tasks requires understanding their structural properties. The *LLM profiler* builds structured profiles to guide offloading decisions from the task prompt and worker metadata. It captures the decomposability characteristics of tasks for parallel partitioning as a feature vector. The vector holds features such as evidence locality, defined by whether the information needed for reasoning and generating answers is localized or global, along with dependency strength across subparts, cross-boundary coupling, and the reasoning horizon that would make parallel execution difficult. The feature vector guides the selection of decomposition axes, such as time or space, and is later used by the *LLM offloading planner* to select a strategy. To profile devices (workers), the profiler estimates each worker's device compute capacity and model performance from available metadata, and normalizes them into comparable relative scores. It uses device identifiers such as CPU model name and deployed model name, leveraging LLM background knowledge about common device classes and model families [25]. It summarizes these estimates as normalized scores in [0,1]: a *device-capacity score* and

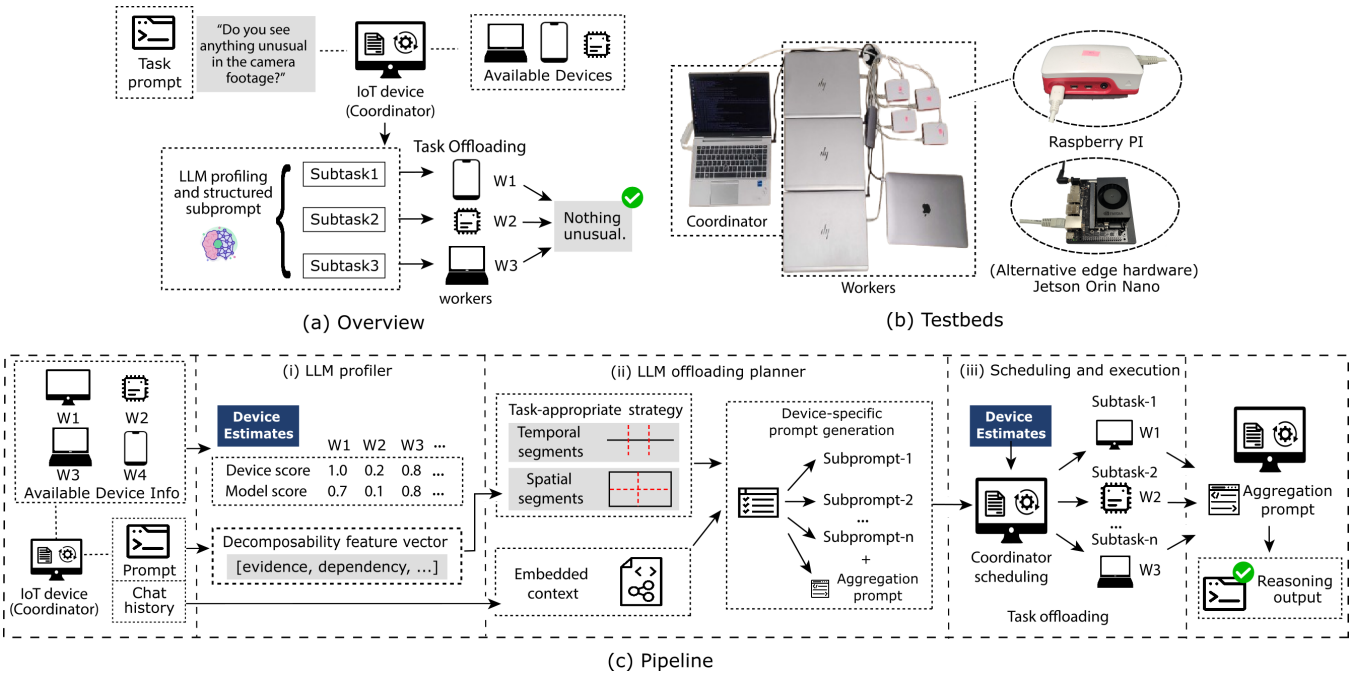


Figure 3: LLMOffload for distributed LLM on-device reasoning; a) LLMOffload overview; b) Testbed with heterogeneous devices. c) LLMOffload overall pipeline;

a *model-capacity score*, which will be used for optimized subtask allocation by the coordinator's scheduler. By using an LLM as a capacity estimator, it can opportunistically exploit whatever metadata is available, even when curated hardware databases are missing or out of date.

Phase II-LLM offloading planner: The *planner* uses the profiler output to select an offloading strategy and generate context-embedded, device-specific subtask prompts and an aggregation prompt. For strategy selection, the planner relies on the task feature vector produced by the profiler, which captures properties such as evidence locality, reasoning horizon, and cross-subtask dependency. We implement two offloading strategies: *Temporal-Segment Offloading* TSO and *Spatial-Segment Offloading* SSO. When the profile suggests a short reasoning horizon and low cross-boundary dependency, the planner selects the strategy *TSO* that partitions the workload along a chronological order of the input. Whereas when the profile indicates that evidence is highly spatially localized, the strategy *SSO* is to partition the input into independent regions or shards and assign workers to analyze specific partitions, focusing computation on the most relevant portions of the input and avoiding unnecessary global processing. After deciding the appropriate strategy, the planner generates device-specific subtask prompts and an aggregation prompt in JSON format. Each subtask prompt embeds information about the specific subtask for the worker (i.e., the work-assignment context), a brief restatement of the subtask objective, and an output contract. Encoding this in the subtask prompt maintains consistency and facilitates output aggregation across workers.

Phase III-Coordinator scheduling and execution: The coordinator receives the planner outputs and leverages decomposition strategy for task assignment to workers. It implements a lightweight, algorithmic scheduler to select workers and allocate work, enabling fast, stable, and predictable decisions. The coordinator can also execute subtasks locally when appropriate. In our prototype, it dispatches assigned subtasks to workers via HTTP POST requests. Each offloading strategy defines a decomposition rule that splits the request into discrete work units; scheduling then assigns these units to workers. We use L to denote the total number of

Table 1: Example prompts and responses from LLMOffload. Bold text indicates the embedded context.

Turn 1: Main task prompt Q. Do you see anything unusual in the camera footage?
Turn 1: Subtask prompt (Worker 1) *Attached clip: 00:00–05:00 of the original footage.* **This is a fixed-position CCTV camera monitoring a restricted-access area.** Analyze only this provided clip to identify any unusual event.
Turn 1: Subtask prompt (Worker 2) *Attached clip: 05:00–10:00 of the original footage.* **This is a fixed-position CCTV camera monitoring a restricted-access area.** Analyze only this provided clip to identify any unusual event.
Turn 1: Aggregation prompt (coordinator) Merge time-aligned evidence from workers, resolve disagreements, and produce one final response.
Turn 2: Follow-up question Q. Who is involved?
Turn 2: Subtask prompt (Worker 1) *Attached clip: 04:10–4:30 of the original footage.* **An unusual event occurred: entry into a restricted area.** Identify the key person(s) involved and describe them using distinguishing visual attributes.

strategy-defined units. Using the per-worker *model-capacity score* and device identifiers, the scheduler first filters out workers that are unreliable or likely to degrade output quality. Then it performs load balancing by distributing the L units across the remaining workers in proportion to each worker's *device-capacity score* defined by s. The set of workers available after filtering is represented by W in Equation 1 and x_i represents the number of units assigned to worker i. The scheduler computes an allocation by approximately solving the following relaxation:

$$\min_{\{x_i\}} \max_{i \in W} \frac{x_i}{s_i} \quad \text{s.t.} \quad \sum_{i \in W} x_i = L, \;\; x_i \geq 0. \tag{1}$$

The optimum equalizes $\frac{x_i}{s_i}$ across workers, yielding $x_i \propto s_i$. The coordinator then rounds $\{x_i\}$ to integers and applies a lightweight adjustment to satisfy practical constraints such as per-worker concurrency limits.

4 THE EXPERIMENTS

We evaluate LLMOffload through systematic experiments across diverse computing environments. First, we assess its ability to decompose LLM reasoning tasks for efficient use of worker devices. We also explore opportunistic use of remote cloud resources, enabling a hybrid approach where tasks can migrate from edge to cloud when beneficial, leveraging more advanced and faster LLMs to reduce the resource burden on cooperative devices.

Dataset: We evaluate LLMOffload on video-based question answering (video QA) workloads using MVBench [29], multiple-choice benchmark over videos covering 20 task types. Examples are illustrated in Table 2. We choose video QA because processing long video inputs and its reasoning are compute-intensive, making offloading a typical and relevant setting. MVBench provides a reproducible test protocol with objective grading across diverse task types and difficulty levels.

Workload construction: We use two workloads. The first consists of original MVBench questions from the two task types that are most sensitive to the offloading strategy. To make that workload, we run a small pilot using Qwen3-VL-8B-Thinking and evaluate both TSO and SSO on 10 questions over all 20 task type. *object_interaction*, where TSO performs best, and *moving_direction*, where SSO performs best are selected. In TSO, we segment the video file into multiple time windows, while in SSO we segment the video to multiple tiles of the frame. To mitigate boundary effects where relevant evidence straddles segment boundaries, overlaps are introduced in both strategies: adjacent temporal segments overlap in *TSO*, and adjacent spatial tiles overlap in *SSO*.

The second workload augments each original question with a context-dependent follow-up over the same video. We construct the follow-up by rewriting a question from a different task type over the same video. We replace the object mention with a pronoun (e.g., it), rendering the follow-up ambiguous without the preceding answer. This evaluates whether the planner carries the necessary context. Table 2 shows example of the workloads. For both workloads, following the standard benchmark setting, we uniformly sample video frames at 2 FPS [1]. The selected videos span durations of 23.45–47.68 seconds (median 32.74 seconds).

Table 2: Examples of questions from our workload.

Question	Candidate	Answer
Workload I		
`object interaction`		
Which object was taken by the person?	The dish. / The box. The blanket. / The paper. The notebook.	The blanket.
`moving direction`		
What direction is the cyan sphere moving in within the video?	The object is stationary. Up and to the right. Down and to the left. Down and to the right.	The object is stationary.
Workload II		
`object interaction (original)`		
Which object was sat at by the person?	The bed. / The sofa. The couch. / The table.	The table.
`action prediction (follow-up)`		
What will the person do next with **it (the table)**?	Take. / Tidy up. Sit at. / Wash.	Sit at.
`moving direction (original)`		
In which direction does the yellow cylinder move in the video?	Up and to the left. Down and to the left. The object is stationary. Up and to the right.	Down and to the left.
`counterfactual inference (follow-up)`		
Without **it (the yellow cylinder)**, which of the following will happen?	cube–red collide gray sphere–red collide green sphere–gray collide cube–green sphere collide	green sphere–gray collide

Testbed: As shown in Figure 3(c), we implement LLMOffload as lightweight coordinator and worker services that communicate over HTTP. Our testbed consists of one coordinator node (HP EliteBook 840 G7, 11th Gen Intel Core i5, 16GB RAM) and a pool of heterogeneous worker devices connected via a local LAN, including three laptop workers (HP EliteBook 840 G6, 8th Gen Intel Core i5), an Apple MacBook Air (M1), and four Raspberry Pi 4 Model B. We also include a NVIDIA Jetson Orin Nano as a worker to analyze GPU acceleration benefits. Each device runs a worker service that uses a local LLM via Ollama [40] with quantized models. The coordinator uses Qwen3-1.7B-Thinking for profiling, planning, and aggregation, as it is among the smallest models that reliably produces our structured, rule-based outputs. For video analysis on workers, we use a mix of locally hosted vision–language models (VLMs) to improve generality, including Qwen3-VL-2B-Instruct, Qwen3-VL-8B-Instruct, and Gemma3-4B.

Evaluation metrics: We measure three metrics: latency, energy consumption, and accuracy. Latency is the end-to-end time per query, measured from when the coordinator gets the request to when it outputs the aggregated answer. Energy is measured on the coordinator using on-chip RAPL counters. Accuracy is MVBench multiple-choice accuracy over the evaluated queries. Together, these metrics capture the core offloading trade-offs: reducing completion time and energy cost without degrading output quality.

Experiment procedure: We evaluate LLMOffload in a controlled setting to establish a clear baseline for offloading behavior. To minimize variability, the experiment uses three identical laptop workers (HP EliteBook 840 G6, 8th Gen Intel Core i5) and fixes the VLM to Qwen3-VL-8B-Instruct. In this setup, the coordinator can

serves as a worker when executing subtasks. For comparison, a local-only baseline runs all computation on the coordinator without offloading under identical inference settings. Scalability is studied by varying the number of active workers from 1 to 3. We further generalize LLMOffload in a practical heterogeneous setting. The worker pool includes one HP laptop, one MacBook Air (M1), and two Raspberry Pi 4 Model B devices. Each device runs a different VLM: the HP laptop runs Qwen3-VL-8B-Instruct, the MacBook Air (M1) runs Gemma3-4B, and the four Raspberry Pi 4 devices run Qwen3-VL-2B-Instruct. Thus, the coordinator does not serve as a worker for these subtasks.

As an ablation study, we also evaluate individual components of LLMOffload. To evaluate the *LLM profiler*'s estimation accuracy, We build a heterogeneous pool of 10 devices and multiple VLMs by randomly sampling from public benchmark sources. Device candidates are drawn from openbenchmarking.org [26], and model candidates use MMLU-Pro [49] scores reported by llm-stats.com [35]. In each run, the profiler estimates the capacity of all candidates, and we validate these estimates against the corresponding benchmark scores. To evaluate the *LLM planner*, for each of the 20 MVBench task types, the planner selects between TSO and SSO. We measure its performance by comparing the chosen strategy against the ground-truth best strategy for each task type. We also evaluate context-embedding performance using the second workload, which includes context-dependent follow-up questions. We compare LLMOffload against three methods: (i) original question (baseline) (ii) naively passing the full conversation history and (iii) providing no extra context.

5 RESULT

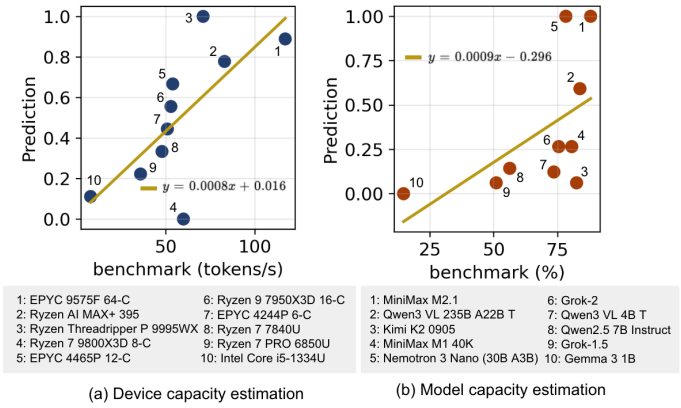
We first validate the effectiveness of LLMOffload's core components: the *LLM profiler* accurately estimates device and model capacity, and the *LLM planner* selects appropriate offloading strategies while embedding the required task context. We then evaluate LLMOffload end to end in both controlled and practical deployment settings.

5.1 LLMOffload component analysis

LLM profiler estimation: Figure 4 shows the alignment between *LLM profiler*'s estimation and benchmark scores. The estimates are generated using Qwen3-1.7B. For *device capacity*, we observe a Pearson correlation of $r = 0.712$ and strong rank agreement with Spearman $\rho = 0.709$. For *model quality*, Pearson correlation is lower ($r = 0.553$), while rank agreement remains high ($\rho = 0.691$). The consistently high Spearman values indicate that the profiler largely preserves the relative ordering across candidates, rather than being driven by a few extreme high- or low-performing devices, and avoids misordering among mid-tier candidates. For *model quality*, the profiler yields broadly consistent rankings across model families (e.g., GPT, Gemini, Claude, and Qwen), though it may exhibit calibration bias that underestimates some mid-range models. Table 3 extends this analysis to other local LLM evaluators and shows that correlation depends on the evaluator choice, ranging from strong agreement to weak or degenerate behavior (reported as N/A). Overall, the *LLM profiler* tracks both device capacity and model quality reasonably well, with especially reliable rank consistency under Qwen3-1.7B.

Table 3: Correlation of capacity estimates across LLMs. Pearson's r and Spearman's ρ between *LLM profiler* estimates and benchmark results.

Model	Device-capacity (r/ρ)	Model-quality (r/ρ)
Qwen3-0.6B	0.269 / 0.059	0.140 / -0.219
Qwen3-1.7B	**0.712** / **0.709**	0.553 / **0.691**
Qwen3-4B	0.687 / 0.636	**0.630** / 0.648
Llama3.2-1B	N/A	N/A
Llama3.2-3B	0.471 / 0.475	0.530 / 0.590



(a) Device capacity estimation (b) Model capacity estimation

Figure 4: Comparing *LLM Profiler* estimates against benchmark results.

Figure 5 shows that the *LLM planner* selects near-optimal offloading strategies. During the construction of WorkloadI, we measure the accuracy each strategy achieves on every task. Using these results, we construct an oracle that always selects the best strategy for each task. In Figure 5(a), the *LLM planner* attains 57.5% accuracy, close to the oracle upper bound of 58.0%. By contrast, random selection achieves 50.75%, corresponding to a quality regret of 7.25 points (vs. 0.5 points for the *LLM planner*), while consistently using a single fixed strategy performs worse, with quality regrets of 3.5 (always TSO) and 11.0 points (always SSO). These results indicate that the optimal offloading strategy is task-dependent and that mismatched strategies can substantially reduce accuracy, making per-task selection non-trivial to decide manually; the LLM planner addresses this challenge by profiling tasks and choosing a more suitable strategy, thereby improving accuracy without sacrificing performance.

LLM planner (embedded context): Figure 6 evaluates the planner's context-embedding mechanism using follow-up questions. In these queries, a key entity from the preceding question is replaced with "it," making the follow-up ambiguous without prior context. On the original questions, accuracy is 50%. With the entity replaced by "it," embedded context preserves the same 50% accuracy. Without context, the system cannot resolve the reference and returns `UNKNOWN`, yielding 0% accuracy. This demonstrates that context embedding is essential for resolving anaphoric references in multi-turn interactions, enabling the planner to interpret follow-ups correctly and avoid complete failure. Naively passing the full chat history

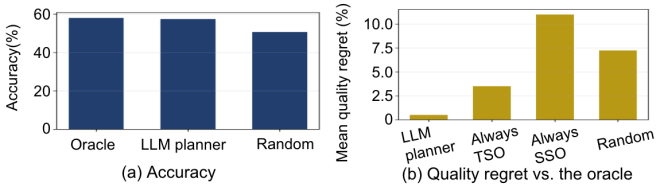
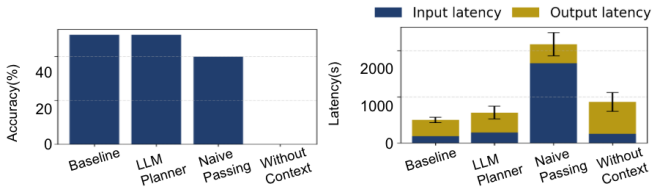


Figure 5: Quality comparison of *LLM planner*-selected offloading strategies



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

from the coordinator to the workers can partially recover context (40% accuracy). However, this approach incurs a substantial latency overhead: loading the entire conversation makes the input roughly 10× longer, increasing prompt processing time and planner latency (Figure 6(b)).

5.2 LLMOffload performance

LLMOffload working flow: Since we have demonstrated the effectiveness of LLMOffload's components, we also describe its end-to-end workflow and the role of each component. Table 4 summarizes the decision traces of the *LLM profiler* and *LLM planner* in the controlled setting as an example, including their rationales, worker outputs, and how the coordinator aggregates responses. The profiler first constructs a task feature vector, indicating cross-boundary = 0 and high requirements for identifiability, local tracking, and global context (all = 2). It also estimates device and model capacity; since we use homogeneous devices and models, all capacities are set to 1.0. Based on these features, the planner selects TSO because the task requires full-scene context to identify the object and temporal cues to locate the pick-up event, while involving no cross-boundary interactions. The planner then generates subtask and aggregation prompts: each subtask asks which object the person takes, provides candidate objects, and instructs the worker to analyze an assigned time window and return a JSON-formatted output for aggregation. This question already includes the full context needed to answer it, so the *LLM planner* does not need to embed additional context though it provides brief situational framing such as assigned time range. Finally, the coordinator aggregates worker outputs via majority voting; workers consistently report interaction with a red bag and identify "bag" with high confidence, so the aggregator outputs "bag," matching the ground truth. Overall, workers follow the prompts consistently and collaborate effectively to solve the task.

Performance in control: We first compare three main metrics under local execution and offloading. Figure 7 summarizes LLMOffload's end-to-end benefits as we scale the number of active workers. Figure 7(a) shows that offloading reduces end-to-end latency for both task types by about 50%. For *object interaction*, latency drops from 27085 s (local) to 13723 s with three workers. For *moving direction*, latency decreases from 26385 s (local) to 12494 s with three workers. Figure 7(b) shows that offloading reduces the coordinator's energy consumption about 65%. For *object interaction*, energy decreases from 137.83 Wh (local) to 48.48 Wh with offloading, and for *moving direction*, it drops from 134.26 Wh to 47.08 Wh. Figure 7(c) shows that offloading does not degrade performance. For *object interaction*, accuracy increases from 60% (local) to 90% with three workers. For *moving direction*, accuracy briefly drops to 50% with two workers but recovers to 60% with three workers, matching local execution. The accuracy gain in object interaction is likely due to time-segment partitioning, which limits each subtask to a shorter, more self-contained window and reduces interference from irrelevant long-context attention [8]. Overall, these results show that LLMOffload can reduce coordinator energy and end-to-end latency without sacrificing accuracy.

The energy savings primarily come from reducing the time spent in VLM inference, and they outweigh the additional overhead introduced by the profiler/planner LLM. On average, the system consumed 2.86 W when idle, 21.94 W while running the profiler/planner LLM, and 18.32 W while running the VLM. By dividing the task across three workers, the VLM inference time decreases from 27 085 s to 11 423 s, with the additional profiling and planning overhead being about 2 300 s, which is substantially smaller than the VLM time reduction. In our current setup, the coordinator also runs a part of the VLM inference after profiling and planning. However, LLMOffload can fully offload VLM execution. Doing so would further reduce coordinator energy from about 130 Wh to about 20 Wh, yielding an 85% reduction.

Inference latency and efficiency: We next analyze the overhead introduced by LLMOffload. Figure 8 breaks down latency by step: on average, profiling takes 841.2 s, planning 635.3 s, and aggregation 620.0 s; 90% of runs complete within 904 s, 825 s, and 701 s, respectively. Network transfer latency is negligible (below 0.01%). Planning exhibits higher variance than profiling and aggregation, likely because it must select an offloading strategy and generate both subtask and aggregation prompts based on task features and estimated model performance, whereas profiling and aggregation operate on more fixed inputs and follow a more standardized workflow. Table 5 reports token usage by step: profiling has the highest average consumption (1446 tokens), followed by planning (1253) and aggregation (841), including *thinking* tokens for Qwen3-1.7B-Thinking. Overall, the pipeline takes 1288 input tokens and generates 3541 output tokens on average. The large variance, especially the occasional near-doubling at the maximum, likely reflects cases

Table 4: Example decision trace of the *LLM profiler / planner*.

Stage	Item	Content
Input	Task	*object_interaction*
	Query (Candidates)	Q: Which object was taken by the person? (A: laptop / B: towel / C: book / D: bag)
Profiler	Task Feature Vector	*{identifiability:2, local_tracking:2, reasoning_horizon:1, global_context:2, small_object:1, cross_boundary:0}*
	Device / Model capacity	*device_capacity: {worker1: 1.0, worker2: 1.0, worker3: 1.0, worker4: 1.0}, model_capacity: {worker1: 1.0, worker2: 1.0, worker3: 1.0, worker4: 1.0}*
Planner	Strategy	**TSO** (temporal segmentation)
	Subtask prompt	You are reviewing frames from an assigned time range ([0%, 25%) / [25%,50%) / [50%,75%) / [75%,100%)). Within this segment only, determine whether the person is taking (i.e., actively picking up / removing / acquiring from a prior resting place) any object below — not merely holding/carrying it. Objects (report exactly one letter if taking occurs): A: laptop, B: towel, C: book, D: bag. Output (exactly one line): If taking occurs: LETTER, confidence=0-1, evidence="..." If no taking can be confirmed: UNKNOWN
	Aggregation prompt	Based on all workers' segment outputs, determine the final object taken. Apply majority voting. If all workers output *UNKNOWN*, output *UNKNOWN*. Do not use any context beyond the workers' segments.
Output	Subtask outputs	Worker 1: {*UNKNOWN(0.0)*} Worker 2: {*D(0.9)* evidence: handling a bag} Worker 3: {*D(0.9)* evidence: handling a red bag} Worker 4: {*D(0.9)* evidence: interacting with a red bag}
	Aggregated result	*final_choice=D, used_segments=[1,2,3].*

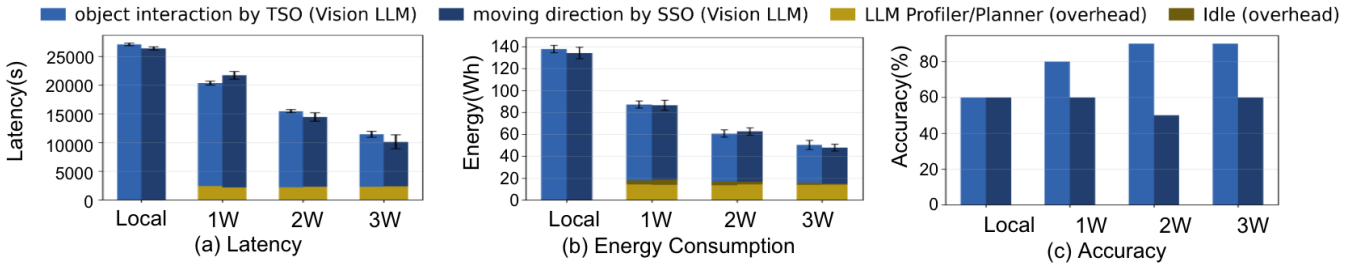


Figure 7: LLMOffload performance across tasks and number of workers. Energy is measured at the coordinator, which also executes subtasks as a worker.

Table 5: Token consumption split into input and output across stages

Stage	Input (min / avg / max)	Output (min / avg / max)
Profiling	407 / 420.1 / 496	1036 / 1446.8 / 2349
Planning	287 / 310.1 / 623	826 / 1253.3 / 2973
Aggregating	539 / 558.0 / 598	473 / 841.0 / 1302
Total	1375 / 1288.2 / 1601	2760 / 3541.1 / 5958

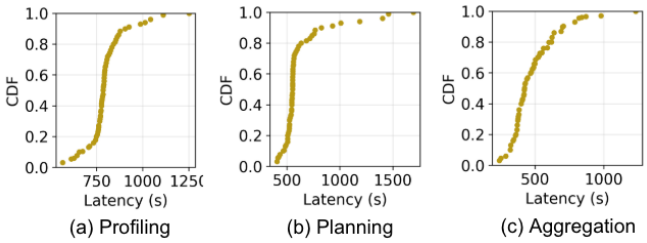


Figure 8: Cumulative distribution functions (CDFs) of coordinator-side LLM process overheads

where the model produces unusually long or failed to produce the end token.

Practical performance: In the practical setting with heterogeneous devices, we evaluated LLMOffload on the same tasks as in the controlled setting. The *LLM profiler* estimated device capacity as HP EliteBook 840 G6: 15, M1 MacBook Air: 20, Raspberry Pi 4 Model B: 1.5, broadly matching benchmark scores (23.1, 73.1, and 4.2, respectively). It also estimated model quality as (Qwen3-VL-8B-Instruct: 95; Gemma3-4B: 80; Qwen3-VL-2B: 75), which was roughly consistent with MVBench results (80, 50, and 60, respectively).

Despite underestimating the M1 MacBook Air and Gemma3-4B, LLMOffload remained effective in practice. Compared with local execution, end-to-end latency decreased by 24.6%, energy consumption decreased by 79.5%, and accuracy remained unchanged at 60%. The larger energy reduction is expected because the coordinator does not act as a worker in this setting. LLMOffload assigns workload in proportion to estimated capacities, so the M1 MacBook Air received about 33% more work than the HP EliteBook; however, we still observed idle time because the MacBook's performance was underestimated, leading to conservative scheduling. Overall,

these results show that LLMOffload can robustly profile heterogeneous resources and achieve substantial latency and energy savings without sacrificing task quality output.

GPU acceleration: We further evaluate the effect of GPU acceleration with a NVIDIA Jetson Orin Nano. In terms of processing speed, the Jetson achieves 11.77 tokens/s for decoding, which is 2.52× faster than the HP EliteBook and 15.49× faster than the Raspberry Pi 4. It also achieves 112.39 tokens/s for prompt processing, corresponding to 1.69× and 10.38× speedups over the HP EliteBook and Raspberry Pi 4, respectively. Overall, these improvements translate into 2.33× and 14.22× end-to-end speedups over the HP EliteBook and Raspberry Pi 4. This makes the on-device LLM reasoning more practical for real-world applications.

However, effectively exploiting client selection requires more accurate characterization and profiling of participating devices. For instance, the current profiler underestimates the Jetson's capabilities. It assigns capacity scores of 0.75 to the Jetson Orin Nano and 1.0 to the HP EliteBook, leaving the Jetson underutilized: it finishes in 502.9 s, while the HP EliteBook becomes the bottleneck at 3207 s. Despite this suboptimal selection, GPU acceleration significantly improves LLM task partitioning in LLMOffload. Compared with conventional multi-agent execution, LLMOffload incurs additional overhead from repeatedly encoding intermediate small language model outputs into prompts on each device. The Jetson's GPU reduces this prompt-encoding latency, mitigating LLMOffload's primary overhead.

6 LLMOFFLOAD EVALUATION IN THE CLOUD

We further evaluate LLMOffload in a cloud environment to assess its generalizability beyond our local, distributed, and resource-constrained heterogeneous testbed.

Experiment setup: We used cloud GPU VMs equipped with NVIDIA H100 SXM GPUs (80 GB VRAM) and accessed all instances via SSH. We chose H100 because it is widely deployed for cloud inference in production services, and is commonly used to host large models at scale. On each VM, we ran Ollama to expose the deployed models as local inference endpoints. We deployed LLMOffload with one coordinator VM and three worker VMs in the same region, using private networking for coordinator–worker communication. The round-trip latency was about 0.8 ms.

We first run offloading experiments using a Qwen3-14B *LLM profiler / planner* and Qwen3-VL-40B workers using 20 questions on the MVBench *moving_direction* task, reporting end-to-end latency and accuracy. In this setup, the coordinator also serves as a worker when executing subtasks. We then vary the models among Qwen3-14B-Thinking, Qwen3-30B-Thinking, gpt-oss-20B, and DeepSeek-R1-32B for the coordinator. On the workers, we deploy three VLMs: Qwen3-VL-30B-Instruct, Mistral-Small-3.2-24B, and Gemma3-27B. These configurations allowed us to study how model choice affects latency and to identify when offloading is beneficial, since gains require the reduction in worker-side workload to outweigh the planning overhead. Throughout, we aimed to reduce energy consumption without sacrificing latency or accuracy. All model choices were constrained by the 80 GB GPU memory of each H100 VM, which prevented us from running larger models.

Table 6: Latency summary for coordinator-side LLM and worker-side VLM on GPU-based cloud environment.

Coordinator LLM		Worker VLM	
Model	**Avg ± Std (s)**	**Model**	**Avg ± Std (s)**
Qwen3-14B-T	20.9 ± 9.8	Qwen3-VL-40B	266.8 ± 109.6
gpt-oss-20B	12.8 ± 3.7	Mistral-S3.2-24B	121.7 ± 67.0
DeepSeek-R1-32B	16.4 ± 2.8	Gemma3-27B	288.4 ± 113.2
Qwen3-30B-T	330.9 ± 53.3		

Table 7: Overhead ratio $f = T_{\text{coord}}/T_{\text{worker}}$ (avg/avg, %). Each cell corresponds to coordinator LLM–worker VLM

Model	Qwen3-VL-40B	Mistral-S3.2-24B	Gemma3-27B
Qwen3-14B-T	7.83%	17.17%	7.25%
gpt-oss-20B	4.80%	10.52%	4.44%
DeepSeek-R1-32B	6.15%	13.48%	5.68%
Qwen3-30B-T	124.03%	271.90%	114.74%

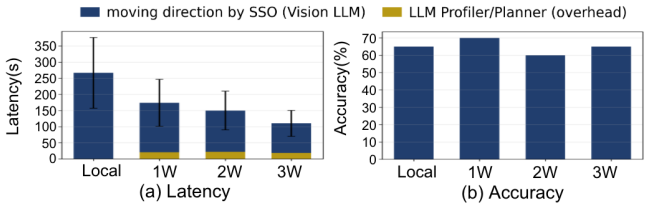


Figure 9: LLMOffload in the cloud environment.

Result: Figure 9 summarizes the cloud results. Overall, the cloud deployment showed the same qualitative trend as the local testbed: offloading reduced end-to-end latency by 58.8% (from 267 s to 110 s) while maintaining MVBench accuracy, and adding more workers further improved latency. The coordination overhead accounted for only 7.8% of the total runtime (about 20 s), so offloading to workers was effective because the reduction in VLM execution time outweighed the added overhead. End-to-end latency showed higher variation than in the local testbed because both the LLM and VLM inference times were more variable in the cloud environment. Since the per-task workload was relatively small compared to the available compute, runtime jitter had a larger impact on total latency. Despite this, the average results followed the same trend.

Table 6 reports latency for different coordinator LLMs and worker VLMs. Coordinator-side profiling and planning took on the order of 10–20 s for most LLMs, whereas worker-side VLM inference ranged from roughly 120 s to 300 s. VLM latency also showed high variability, with the standard deviation around 30% of the mean, which contributed to higher end-to-end latency variation. Table 7 reports the overhead ratio. With well-matched coordinator and worker models, the overhead remained around 5–15%, making offloading beneficial because the VLM time saved outweighed the coordination cost. In contrast, using Qwen3-30B-Thinking on the coordinator led to overhead larger than the VLM inference time, in which case offloading was not worthwhile. Overall, these measurements explain why the cloud results match the local trend: offloading works best when worker-side VLM inference dominates the runtime.

7 RELATED WORK

Computational offloading: Research on computational offloading has evolved significantly over the past two decades. Task offloading has traditionally been associated with computational execution; however, its scope has expanded to include other operations, including sensing and networking [12–14]. Early work on cyber-foraging demonstrated methods for offloading computationally intensive mobile operations to nearby cloudlets [27, 30, 61], followed by cloud-based approaches in which heavy routines were executed remotely on large-scale infrastructure [4, 43]. Subsequent studies examined decision factors such as latency [3, 46], bandwidth variability [51, 58], application requirements [10, 55], and device energy constraints [33, 63]. With the emergence of edge computing, hybrid offloading strategies that combine edge and cloud resources have been explored [33, 38]. Similarly, some works have explored distributing tasks among personal and wearable devices to reduce the processing load of individual devices when facing resource-intensive tasks [9, 60]. Offloading has also been extended to DNN inference, enabling intensive operations on constrained devices. For instance, offloading strategies have been applied for layer-wise DNN partitioning to manage latency across mobile-edge-cloud infrastructures [16, 19, 28]. LLM task offloading has also been explored, but existing approaches typically treat inference as a monolithic process that follows a fixed computational graph [57]. In contrast, our work decomposes the reasoning process into distinct subtasks that can be offloaded independently. Thus, *LLMOffload recognizes on-device LLM inference as a sequence of reasoning steps and enables these steps to be selectively offloaded across heterogeneous environments for distributed and efficient execution.*

Distributed and cooperative devices: Collaboration between devices has traditionally been explored through master–slave architectures [15]. More recently, collaboration across heterogeneous models has emerged in the context of LLM training [18, 45, 62]. Distributed prompting enables the modular decomposition of complex instructions into manageable units that can be delegated to and handled by multiple coordinated models [22, 42]. This paradigm has been further explored through schedulers that decompose LLM tasks within the same runtime environment or shared memory [59, 62]. While these approaches highlight the benefits of dividing reasoning across multiple agents, they typically assume cloud-based settings with homogeneous compute resources and overlook heterogeneous edge environments with strict latency and energy constraints [21, 36]. Moreover, existing systems do not incorporate device state or quality constraints into the task delegation process. *LLMOffload contributes a pipeline that enables task division and execution while preserving logical continuity and coherence, and is explicitly designed for heterogeneous, distributed environments tailored to resource-constrained devices.*

LLMs reasoning and agent planning: LLMs emulate human-like reasoning by generating structured, sequential intermediate steps that reveal how a response is produced, commonly referred to as Chain-of-Thought (CoT) [50]. Recent advances in LLM planning and CoT have shifted inference from monolithic, sequential generation to modular, multi-step workflows that are explicitly explorable [6, 20, 54], enabling deliberate search for optimal reasoning paths [2, 56]. While CoT increases the computational density of individual reasoning steps, planning organizes these steps into structured dependencies, exposing opportunities for improved resource efficiency [23, 24, 41]. Distributed prompting exploits this modularity by delegating sub-tasks across multiple models or agents to improve accuracy. Existing work typically distributes and plans task execution within homogeneous environments, where multiple agents collaborate on different parts of the reasoning process, for example, agent planning entirely within cloud infrastructures. In contrast, LLMOffload enables distributed reasoning across heterogeneous environments by offloading subtasks to different AI agents and selecting the most suitable devices for execution. Specifically, *LLMOffload overcomes the limitations of homogeneous planning by allowing heterogeneous devices to execute LLM subtasks through structured prompting and shared context windows. In addition, device specific characteristics, such as energy availability, latency, and CPU capacity, are encoded as semantic descriptors to improve device profiling and optimize task allocation and execution.*

8 DISCUSSION

Room for improvement: LLMOffload demonstrates strong generalizability across tasks but also reveals important trade-offs. Smaller on-device models reduce overhead but limit reasoning reliability, motivating hybrid designs that combine lightweight LLMs with fallback heuristics. While the current decomposition strategies are effective, extending them beyond specific workloads would broaden their applicability. Distributed execution also creates opportunities for optimization through profiling, caching, and the reuse of semantically similar prompts. Furthermore, distributed client selection can improve task execution by assigning more capable devices to computationally intensive tasks. For example, in pervasive camera systems, video frames can be partitioned into regions with different resolutions for continuous tracking, preserving high resolution only in areas with significant changes [52], allowing these regions to be assigned to more capable workers [53]. However, the benefits of offloading depend on keeping coordination overhead low, as energy savings arise primarily from reducing active coordination time rather than inference itself. Finally, future work should explore privacy- and energy-aware task partitioning, selectively offloading computations while keeping sensitive reasoning on-device and jointly optimizing battery lifetime, latency, and reasoning quality.

On-device LLM: Offload or not?: While we demonstrate the feasibility of offloading LLM reasoning across capable devices to reduce energy consumption and end-to-end latency, realizing these gains critically depends on controlling coordination overhead. Contrary to the intuition that preserving output quality necessarily incurs proportional energy costs, our results show that energy savings are primarily driven by reducing the active execution time of coordination. This observation aligns with classical code offloading principles, where offloading is beneficial only when computation intensity outweighs communication and coordination costs. In our setting, offloading yields substantial benefits when coordination overhead remains significantly smaller than worker inference time, as shown in Figure 9. However, current on-device LLMs remain less mature than their cloud counterparts, and resource-constrained

edge coordinators running smaller models often lack sufficient reasoning capability to generate reliable offloading plans, making it challenging to balance planning quality against coordination overhead in heterogeneous IoT environments.

Cloud-Edge-IoT workloads: We show that LLMOffload enables the distribution and cooperative execution of on-device LLM tasks across heterogeneous devices. When opportunistically available, multiple layers of resources can be leveraged to enhance the reasoning capability of on-device models through edge or cloud support. Achieving transparent task migration across these layers, however, requires coordination within each layer as well as communication across coordinators to negotiate and establish execution plans prior to deployment. A promising direction is the ability to predict resource availability at different layers in order to proactively schedule task execution; we are particularly interested in investigating whether LLMOffload can exploit such predictive capabilities to further improve efficiency and responsiveness.

On LLM partitioning: While small language models often produce less reliable decisions than larger models, their performance can be substantially improved by augmenting them with domain specific knowledge. However, resource constrained devices still introduce reasoning latency, making it important to match task complexity to the computational capabilities of each device. The increasing availability of GPU acceleration at the edge has significantly reduced this latency, but further improvements require distributing inference across multiple GPU enabled devices, motivating solutions such as LLMOffload. As GPU accelerated edge deployments become denser and more widely available (like in fog computing environments), distributed LLM partitioning can emerge as a fundamental approach for executing large language models efficiently across decentralized infrastructure.

On IoT and emerging distributed applications: Distributed inference and training are well suited to sensing applications, where data is generated by multiple distributed sensors. Advances in model compression and distributed ML/DL execution have made such deployments increasingly practical. Although on-device LLMs still exhibit high inference latency for real-time applications such as human activity recognition [32], the growing availability of edge GPU acceleration is reducing this gap by accelerating inference. Our results demonstrate that hardware acceleration significantly improves performance, while further latency reductions will require cooperative distributed inference techniques that leverage the computational resources of multiple edge devices.

9 SUMMARY AND CONCLUSIONS

We presented LLMOffload, an on-device LLM framework that cooperatively offloads reasoning complexity across multiple heterogeneous devices. Unlike prior approaches, LLMOffload enables the decomposition of LLM reasoning tasks in heterogeneous and resource constrained environments. The framework incorporates a profiler and planner that determine whether and how LLM subtasks should be offloaded to nearby devices. Through rigorous and systematic evaluation, our results show that energy savings are primarily driven by reductions in active execution time during coordination, consistent with classical offloading principles in which benefits arise only when computational intensity outweighs communication overhead. We also show that these benefits are further amplified when participating devices provide GPU acceleration.. By effectively exploiting this trade off, LLMOffload enables practical offloading of LLM reasoning to nearby devices. Our work paves the way for cooperative and heterogeneous environments in which multiple devices collaboratively solve complex reasoning tasks.

ACKNOWLEDGMENTS

This work was supported by Estonian Research Council grant (PRG 2725). Special thanks to the reviewers and the shepherd for the valuable suggestions.

REFERENCES

[1] Shuai Bai, Yuxuan Cai, Ruizhe Chen, et al. 2025. Qwen3-VL Technical Report. *arXiv preprint arXiv:2511.21631* (2025).
[2] Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 38. 17682–17690.
[3] Chinmay Chakraborty, Kaushik Mishra, Santosh Kumar Majhi, and Hemanta Kumar Bhuyan. 2022. Intelligent latency-aware tasks prioritization and offloading strategy in distributed fog-cloud of things. *IEEE Transactions on Industrial Informatics* 19, 2 (2022), 2099–2106.
[4] Byung-Gon Chun and Petros Maniatis. 2009. Augmented smartphone applications through clone cloud execution.. In *HotOS*, Vol. 9. 8–11.
[5] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. 2010. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*. 49–62.
[6] Zeyuan Ding, Dian Ding, Han Zhang, Jiannong Cao, and Guangtao Xue. 2025. LLM4Load: An LLM Prompt-Driven Framework for Multi-Scale Workload Prediction. In *2025 IEEE International Conference on Web Services (ICWS)*. IEEE, 293–304.
[7] Luobing Dong, Weili Wu, Qiumin Guo, Meghana N Satpute, Taieb Znati, and Ding Zhu Du. 2019. Reliability-aware offloading and allocation in multilevel edge computing system. *IEEE Transactions on Reliability* 70, 1 (2019), 200–211.
[8] Yufeng Du et al. 2025. Context Length Alone Hurts LLM Performance Despite Perfect Retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2025*. Association for Computational Linguistics, Suzhou, China, 23281–23298. https://doi.org/10.18653/v1/2025.findings-emnlp.1264
[9] Haoyang Fan, Yi-Chien Lin, and Viktor Prasanna. 2025. ELLIE: Energy-Efficient LLM Inference at the Edge Via Prefill-Decode Splitting. In *2025 IEEE 36th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 139–146.
[10] Qiang Fan and Nirwan Ansari. 2018. Application aware workload allocation for edge computing-based IoT. *IEEE Internet of Things Journal* 5, 3 (2018), 2146–2153.
[11] Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. 1987. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 9, 3 (1987), 319–349.
[12] Huber Flores, Pan Hui, Petteri Nurmi, et al. 2018. Evidence-Aware Mobile Computational Offloading. *IEEE Transactions on Mobile Computing* 17, 8 (2018), 1834–1850.
[13] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Srirama, and Rajkumar Buyya. 2015. Mobile code offloading: from concept to practice and beyond. *IEEE Communications Magazine* 53, 3 (2015), 80–88.
[14] Huber Flores, Xiang Su, Vassilis Kostakos, Jukka Riekki, Eemil Lagerspetz, Sasu Tarkoma, Pan Hui, Yong Li, and Jukka Manner. 2017. Modeling mobile code acceleration in the cloud. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 480–491.
[15] Huber Flores, Agustin Zuniga, Farbod Faghihi, Xin Li, Samuli Hemminki, Sasu Tarkoma, Pan Hui, and Petteri Nurmi. 2020. Cosine: Collaborator selector for cooperative multi-device sensing and computing. In *2020 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 1–10.
[16] Mingjin Gao et al. 2021. Task partitioning and offloading in DNN-task enabled mobile edge computing networks. *IEEE Transactions on Mobile Computing* 22, 4 (2021), 2435–2445.
[17] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300* (2020).

[18] Wei Huang, Yinggui Wang, Anda Cheng, Aihui Zhou, Chaofan Yu, and Lei Wang. 2024. A fast, performant, secure distributed training framework for llm. In *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 4800–4804.

[19] Hyuk-Jin Jeong, Hyeon-Jae Lee, Kwang Yong Shin, Yong Hwan Yoo, and Soo-Mook Moon. 2020. PerDNN: Offloading deep neural network computations to pervasive edge servers. In *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1055–1066.

[20] Yongjian Jia, Jianping Cai, and Ximeng Liu. 2025. Entropy-Guided Tree of Thoughts: A Dynamic Approach to Diverse Path Generation in LLM Reasoning. In *2025 3rd International Conference on Big Data and Privacy Computing (BDPC)*. IEEE, 132–136.

[21] Hongpeng Jin and Yanzhao Wu. 2025. Ce-collm: Efficient and adaptive large language models through cloud-edge collaboration. In *2025 IEEE International Conference on Web Services (ICWS)*. IEEE, 316–323.

[22] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406* (2022).

[23] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2024. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*.

[24] İbrahim Kök, Orhan Demirci, and Suat Özdemir. 2024. When iot meet llms: Applications and challenges. In *2024 IEEE International Conference on Big Data (BigData)*. IEEE, 7075–7084.

[25] Reo Kuchida et al. 2025. Assembling Edge Intelligence and Decentralized Infrastructure on Command using LLMs. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications Workshops*. 1–6.

[26] Michael Larabel. 2025. *Llama.cpp Benchmark (pts/llama-cpp) — OpenBenchmarking.org*. OpenBenchmarking.org (Phoronix Media). https://openbenchmarking.org/test/pts/llama-cpp Last updated: 2025-11-16. Accessed: 2026-01-20.

[27] Grace A Lewis, Sebastian Echeverría, Soumya Simanta, Ben Bradshaw, and James Root. 2014. Cloudlet-based cyber-foraging for mobile systems in resource-constrained edge environments. In *Companion Proceedings of the 36th International Conference on Software Engineering*. 412–415.

[28] Hui Li, Xiuhua Li, Qilin Fan, Qiang He, Xiaofei Wang, and Victor CM Leung. 2024. Distributed DNN inference with fine-grained model partitioning in mobile edge computing networks. *IEEE Transactions on Mobile Computing* 23, 10 (2024), 9060–9074.

[29] Kunchang Li, Yali Wang, Yinan He, Yizhuo Li, Yi Wang, Yi Liu, Zun Wang, Jilan Xu, Guo Chen, Ping Luo, et al. 2024. Mvbench: A comprehensive multi-modal video understanding benchmark. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 22195–22206.

[30] Yuanjie Li, Nguyen Tung Anh, Azhar Saeed Nooh, Kuwon Ra, and Minho Jo. 2018. Dynamic mobile cloudlet clustering for fog computing. In *2018 international conference on electronics, information, and communication (iceic)*. IEEE, 1–4.

[31] Ji Lin et al. 2025. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *GetMobile: Mobile Computing and Communications* 28, 4 (2025), 12–17.

[32] Jiashu Liu, Kevin Post, Reo Kuchida, Agustin Zuniga, Fatemeh Sarhaddi, Huber Flores, Petteri Nurmi, and Ngoc Thi Nguyen. 2026. LLMSEG: Semantic Segmentation and Few-Shot Recognition of Human Activity Recognition Data Using Large Language Models. In *Proceedings of the 2026 ACM/IEEE International Conference on Embedded Artificial Intelligence and Sensing Systems*. 1016–1028.

[33] Yu Liu, Yingling Mao, Xiaojun Shang, Zhenhua Liu, and Yuanyuan Yang. 2023. Energy-aware online task offloading and resource allocation for mobile edge computing. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 339–349.

[34] Mohan Liyanage, Farooq Dar, Rajesh Sharma, et al. 2021. GEESE: Edge computing enabled by UAVs. *Pervasive and Mobile Computing* 72 (2021), 101340.

[35] llm-stats.com. 2026. AI Leaderboards 2026. Website. https://llm-stats.com/ Accessed: 2026-01-20.

[36] Haoxiang Luo, Yinqiu Liu, Ruichen Zhang, Jiacheng Wang, Gang Sun, Dusit Niyato, Hongfang Yu, Zehui Xiong, Xianbin Wang, and Xuemin Shen. 2025. Toward edge general intelligence with multiple-large language model (Multi-LLM): architecture, trust, and orchestration. *IEEE Transactions on Cognitive Communications and Networking* (2025).

[37] Thomas J McCabe. 1976. A complexity measure. *IEEE Transactions on software Engineering* 4 (1976), 308–320.

[38] Xianling Meng, Wei Wang, and Zhaoyang Zhang. 2017. Delay-constrained hybrid computation offloading with cloud and fog computing. *IEEE Access* 5 (2017), 21355–21367.

[39] Mohammad Movahedi and Juyeong Choi. 2024. The crossroads of llm and traffic control: A study on large language models in adaptive traffic signal control. *IEEE Transactions on Intelligent Transportation Systems* (2024).

[40] Ollama Inc. 2025. *Ollama*. https://ollama.com/ Accessed: 2026-01-05.

[41] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 118–132.

[42] Ratish Puduppully, Anoop Kunchukuttan, Raj Dabre, Aiti Aw, and Nancy Chen. 2023. DecoMT: Decomposed prompting for machine translation between related languages using large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. 4586–4602.

[43] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* 8, 4 (2009), 14–23.

[44] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. 2015. Edge analytics in the internet of things. *IEEE Pervasive Computing* 14, 2 (2015), 24–31.

[45] Yongliang Shen, Kaitao Song, et al. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems* 36 (2023), 38154–38180.

[46] Xiang Sun and Nirwan Ansari. 2017. Latency aware workload offloading in the cloudlet network. *IEEE Communications Letters* 21, 7 (2017), 1481–1484.

[47] Andrea Tagliabue, Kota Kondo, Tong Zhao, Mason Peterson, Claudius T Tewari, and Jonathan P How. 2024. Real: Resilience and adaptation using large language models on autonomous aerial robots. In *2024 IEEE 63rd Conference on Decision and Control (CDC)*. IEEE, 1539–1546.

[48] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. 2020. A survey on distributed machine learning. *Acm computing surveys (csur)* 53, 2 (2020), 1–33.

[49] Yubo Wang et al. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems* 37 (2024), 95266–95290.

[50] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[51] Rich Wolski, Selim Gurun, Chandra Krintz, and Dan Nurmi. 2008. Using bandwidth data to make computation offloading decisions. In *IEEE international symposium on parallel and distributed processing*. 1–8.

[52] Ji-Yan Wu, Vithurson Subasharan, Tuan Tran, and Archan Misra. 2022. Mrim: Enabling mixed-resolution imaging for low-power pervasive vision tasks. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. IEEE, 44–53.

[53] Ji-Yan Wu, Vithurson Subasharan, Tuan Tran, and Archan Misra. 2022. MRIM: Enabling Mixed-Resolution Imaging for Low-Power Pervasive Vision Tasks. In *2022 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. 44–53. https://doi.org/10.1109/PerCom53586.2022.9762398

[54] Yifei Xia, Fangcheng Fu, Wentao Zhang, Jiawei Jiang, and Bin Cui. 2024. Efficient multi-task llm quantization and serving for multiple lora adapters. *Advances in Neural Information Processing Systems* 37 (2024), 63686–63714.

[55] Han Xiao, Changqiao Xu, Yunxiao Ma, Shujie Yang, Lujie Zhong, and Gabriel-Miro Muntean. 2022. Edge intelligence: A computational task offloading scheme for dependent IoT application. *IEEE Transactions on Wireless Communications* 21, 9 (2022), 7222–7237.

[56] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems* 36 (2023), 11809–11822.

[57] Shengyuan Ye, Bei Ouyang, Liekang Zeng, Tianyi Qian, Xiaowen Chu, Jian Tang, and Xu Chen. 2025. Jupiter: Fast and resource-efficient collaborative inference of generative llms on edge devices. In *IEEE INFOCOM 2025-IEEE Conference on Computer Communications*. IEEE, 1–10.

[58] Lei Zhang, Yuhong Zhong, Jiangchuan Liu, and Laizhong Cui. 2023. Resource and bandwidth-aware video analytics with adaptive offloading. In *2023 IEEE 20th International Conference on Mobile Ad Hoc and Smart Systems (MASS)*. IEEE, 107–115.

[59] Mingjin Zhang, Xiaoming Shen, Jiannong Cao, Zeyang Cui, and Shan Jiang. 2024. Edgeshard: Efficient llm inference via collaborative edge computing. *IEEE Internet of Things Journal* (2024).

[60] Songge Zhang, Wen Wu, Penghui Hu, Shaofeng Li, and Ning Zhang. 2023. Split federated learning: Speed up model training in resource-limited wireless networks. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 985–986.

[61] Yang Zhang, Dusit Niyato, and Ping Wang. 2015. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing* 14, 12 (2015), 2516–2529.

[62] Yusen Zhang, Ruoxi Sun, Yanfei Chen, Tomas Pfister, Rui Zhang, and Sercan Arik. 2024. Chain of agents: Large language models collaborating on long-context tasks. *Advances in Neural Information Processing Systems* 37 (2024), 132208–132237.

[63] Zhenyu Zhang, Huan Zhouand, Liang Zhao, and Victor CM Leung. 2022. Digital twin assisted computation offloading and service caching in mobile edge computing. In *2022 IEEE 42nd international conference on distributed computing systems (ICDCS)*. IEEE, 1296–1297.