BenchCouncil Transactions on Benchmarks, Standards and Evaluations 5 (2025) 100249

Contents lists available at ScienceDirect

BenchCouncil Transactions on Benchmarks, Standards and Evaluations

journal homepage: www.keaipublishing.com/en/journals/benchcouncil-transactions-on-benchmarks-standards-and-evaluations/

Full length article

An evaluation framework for measuring prompt wise metrics for large language models in resource-constrained edge

Partha Pratim Ray *, Mohan Pratap Pradhan

Department of Computer Applications, Sikkim University, Gangtok, Sikkim, India

ARTICLE INFO

Keywords:
Large language model
LLM evaluation
LLM metrics
Evaluation framework
Quantized LLMs
Resource-constrained environment

ABSTRACT

Existing challenges in deploying large language models (LLMs) on resource-constrained devices stem from limited CPU throughput, memory capacity, and power budgets. Motivated by the lack of edge-specific evaluation tools, we introduce **LLMEvaluator**, a framework that profiles quantized LLMs — Qwen2.5, Llama3.2, Smollm2, and Granite3 — on a Raspberry Pi 4B using a suite of core and derived metrics. Our contributions include (i) a unified taxonomy that integrates latency, throughput, power variation, memory stability, and thermal behavior; (ii) prompt-wise analyses across ten NLP tasks; and (iii) correlation studies guiding optimizations. Key results show that Qwen2.5 leads in energy efficiency and throughput with a 68.44 MB memory standard deviation; Granite3 excels in memory stability , minimal load overhead, and per-token latency; Smollm2 suffers the highest total duration, longest prompt overhead, and lowest power efficiency; and Llama3.2 balances latency, throughput (8.12 tokens/s), and energy per token with moderate power variability (1.05 W std dev). Correlation analysis reveals that reducing model load time yields the largest improvement in end-to-end latency ($r > 0.9$), and that throughput gains directly translate into energy savings ($r \approx -0.81$). **LLMEvaluator** empowers selection and tuning of LLMs for low-power environments.

1. Introduction

LLMs have become central to a wide range of applications, from complex natural language processing tasks to conversational AI [1–3]. However, deploying these models on hardware with strict resource limits — such as the Raspberry Pi 4B or other CPU-only platforms — poses significant challenges [4,5]. Specifically, edge devices must contend with high computational and memory requirements, variable power draw, and tight latency constraints [6,7]. Conventional evaluation methods, originally developed for data centers and high-performance clusters, often fail to account for the dynamic power fluctuations and memory ceilings characteristic of edge environments [8–10]. As a result, there is an urgent demand for specialized frameworks that generate actionable performance insights under these localized conditions [11,12].

Simultaneously, the rapid adoption of edge computing brings AI workloads closer to data sources, reducing latency and improving privacy [13,14]. Yet this shift also introduces new obstacles for large-scale models [15–17]. Existing benchmarks typically ignore critical edge-specific factors — limited CPU headroom, constrained onboard memory, and the imperative for energy efficiency — leading to suboptimal deployments and scalability issues on low-power devices [18–21]. To address these shortcomings, a robust evaluation framework tailored to edge requirements is essential, ensuring LLMs remain both practical and efficient under resource constraints [22–26].

The aims of this study are threefold such as (i) to design a framework that evaluates LLMs using metrics specific to edge environments, (ii) to measure and quantify the resource consumption and efficiency of LLMs on constrained hardware, and (iii) to introduce novel metrics to provide a deeper, more actionable performance assessment.

To this end, we present LLMEvaluator, a modular framework that unifies power efficiency, memory utilization, and CPU stability metrics to illuminate the trade-offs between computational throughput and resource usage. Validated on quantized versions of Qwen2.5, Llama3.2, Smollm2, and Granite3 running on a Raspberry Pi 4B, LLMEvaluator demonstrates its flexibility across diverse CPU-only platforms. In addition to traditional throughput and accuracy measures, our framework proposes new metrics that capture energy per token and thermal load behavior during inference, thereby bridging the gap between model-centric and system-centric evaluations. This comprehensive approach enables practitioners to optimize model selection, fine-tune hardware configurations, and enhance the reliability of LLM-based services in edge deployments. LLMEvaluator's potential extends across both industrial and research domains [27–30]. For example, in manufacturing automation, it can assess LLM performance for predictive

* Corresponding author.
E-mail address: ppray@cus.ac.in (P.P. Ray).

https://doi.org/10.1016/j.tbench.2025.100249
Received 14 May 2025; Received in revised form 28 November 2025; Accepted 8 December 2025
Available online 11 December 2025

maintenance under strict power and latency constraints, while in healthcare it can validate real-time translation models on portable diagnostic tools without exceeding available resources.

The major contributions of this work can be summarized as follows:

- An extensible framework that unifies model-centric and system-centric assessments for edge deployments by integrating accuracy metrics with CPU/memory utilization and power-draw variability.
- Introduction of various novel metrics to characterize the complex interactions between LLM computation and constrained hardware resources.
- Implementation of evaluations across diverse task categories — from general knowledge to mathematical reasoning — paired with correlation analyses that expose metric interdependencies and guide targeted optimizations.
- Deployment on a Raspberry Pi 4B using quantized Qwen2.5, Llama3.2, Smollm2, and Granite3 models, uncovering distinct trade-offs: Qwen2.5 leads in throughput and energy efficiency, Granite3 excels in memory stability, and Smollm2 exhibits the highest resource consumption.

The remainder of this paper is structured as follows. Section 2 reviews related work and identifies existing gaps. Section 3 details the architecture and implementation of LLMEvaluator framework. Section 4 describes the curated prompt set for evaluating model behavior across NLP tasks. Section 5 presents the LLMEvaluator algorithm. Section 6 defines our taxonomy of core, resource-aware, and derived metrics. Section 7 showcases experimental results, including correlation and prompt-wise analyses. Section 8 presents elaborative discussion of this work. Finally, Section 9 concludes with future research directions, emphasizing heterogeneous platform support and sustainability metrics in resource-constrained edge.

2. Related works

A number of benchmarks and evaluation frameworks have emerged to assess the capabilities of large language models (LLMs), yet few address the stringent requirements of edge deployments. Wang et al. [31] introduced PandaLM, which emphasizes instruction-tuning quality by measuring aspects such as brevity, clarity, and conformity to prompts. Leveraging human-annotated datasets and a trained "judge" LLM, PandaLM aligns model outputs with user expectations while avoiding external API calls. However, its scope is limited to optimizing hyperparameters and instruction-following in general-purpose LLMs, without considering energy consumption or memory constraints characteristic of edge hardware.

In the domain of process mining, Berti et al. [32] developed PM-LLM-Benchmark to evaluate LLM proficiency on specialized mining tasks. Their study demonstrated that smaller, edge-suitable architectures often underperform on these complex workflows, highlighting the absence of lightweight evaluation tools tailored to constrained environments. Similarly, Zhou et al. [33] proposed ElecBench for power dispatch scenarios, combining six high-level metrics (factuality, logicality, stability, security, fairness, expressiveness) and 24 submetrics. While ElecBench provides a comprehensive view of decision-making quality in the energy sector, it presumes access to abundant computational resources and overlooks edge-specific efficiency requirements.

Multimodal benchmarks have also been put forth, though chiefly for data-center settings. Xu et al. [34] presented LVLM-eHub to assess vision–language models on tasks such as visual question answering and hallucination detection, and Sun et al. [35] introduced SciEval for scientific research applications, employing dynamic subsets to prevent data leakage. Both frameworks deliver rigorous quantitative and qualitative analyses but omit measurements of power draw, thermal behavior, and memory footprint. Mobile-Bench by Deng et al. [36] evaluates LLM-powered mobile agents with novel planning metrics, yet does not prioritize computational optimization. Conversely, Yang et al. [37]'s LLMCBench focuses on compression techniques to reduce model size, contributing valuable insights into efficiency gains but stopping short of examining thermal or power stability.

Additional efforts have explored LLMs as evaluators or in multilingual and planning contexts. Chen et al. [38]'s MLLM-as-a-Judge framework addresses hallucination and bias in multimodal evaluations, while Spangher et al. [39]'s Project MPG merges accuracy and cost into aggregate scores. Son et al. [40]'s MM-Eval covers performance across 18 languages, and Valmeekam et al. [41]'s PlanBench examines planning and reasoning capabilities. In recommendation systems, Liu et al. [42] introduced LLMRec for explainability tasks, and Liu et al. [43] extended safety evaluations with MM-SafetyBench. Chu et al. [44]'s PRE uses peer review to reduce evaluator bias, and Yu et al. [45]'s KoLA benchmarks world-knowledge across 19 tasks with a novel self-contrast metric. Although each contributes important evaluation perspectives, none systematically measure the power, memory, and thermal constraints critical to edge computing. Table 1 compares our proposed work with existing literature.

These prior works exhibit increasing sophistication in LLM assessment but consistently neglect edge-specific concerns such as limited CPU throughput, memory ceilings, and energy efficiency. Our LLMEvaluator framework addresses this gap by targeting resource-constrained environments with a lightweight design, support for quantized LLMs, and CPU-optimized inference. We introduce localized metrics —
including power usage variation, thermal load factor, and sustained inference efficiency — to provide a comprehensive, actionable evaluation tailored to low-power edge platforms.

Limitations of Existing Works

Despite the wealth of LLM benchmarking tools, current frameworks share several critical shortcomings when it comes to edge deployment. First, they assume abundant compute and memory resources, making them unsuitable for CPU-only single-board computers with strict power and thermal budgets. Second, they focus almost exclusively on aggregate throughput or quality scores, ignoring transient behaviors in power draw, memory usage spikes, and prompt-specific latency that dominate real-world edge scenarios. Third, existing benchmarks treat inference as a monolithic operation, failing to separate model loading, prompt processing, and token generation—each of which can present a unique bottleneck on constrained hardware. Finally, there is no facility for prompt-wise analysis across diverse task types, so practitioners cannot tailor model selection or optimization strategies to the particular mix of queries their applications will encounter.

Novelty of LLMEvaluator

LLMEvaluator overcomes these gaps by introducing a fine-grained, resource-aware evaluation methodology expressly designed for low-power, CPU-only environments. It unifies a rich taxonomy of core performance metrics (load time, token throughput), system-level telemetry (real-time CPU, RAM, power sampling), and derived efficiency indices (energy per token, thermal load factor) into a prompt-wise analysis pipeline. By decomposing inference into loading, prompt ingestion, and generation phases and correlating each phase's behavior with dynamic resource usage, LLMEvaluator delivers actionable insights that no existing tool provides. Its modular, extensible design supports quantized model evaluation on single-board computers like the Raspberry Pi 4B — and can be readily ported to other edge platforms — empowering practitioners to make informed trade-offs between latency, energy, and stability in production deployments.

3. LLMEvaluator framework

Fig. 1 illustrates the end-to-end architecture of LLMEvaluator, a purpose-built framework that unites a lightweight web front-end, an asynchronous request dispatcher, an on-device inference engine, and

Table 1
Comparison of existing benchmarks and LLMEvaluator

Benchmark	Year	Domain/Task	Core metrics	Edge-aware	Key Findings/Limitations	Edge support
PandaLM [31]	2023	Instruction tuning	Correctness; brevity; clarity; instruction adherence; comprehensiveness; formality	No	Matches 93.8% of GPT-3.5 and 88.3% of GPT-4 on F1, but omits power and memory considerations.	No
PM-LLM-Benchmark [32]	2024	Process mining	Domain-specific accuracy; implementation strategy success	No	Demonstrates tiny models underperform on PM tasks; does not track resource usage.	No
ElecBench [33]	2024	Power dispatch	Factuality; logicality; stability; security; fairness; expressiveness (6); 24 submetrics	No	Evaluates eight LLMs on sector scenarios; assumes high compute; ignores energy/memory limits.	No
LVLM-eHub [34]	2024	Vision–language multimodal	VQA accuracy; object hallucination rate; user-level arena score	No	Benchmarks 13 LVLMs quantitatively and via online arena; no power or thermal profiling.	No
SciEval [35]	2024	Scientific research	Bloom's taxonomy (4 levels); objective vs subjective Q/A; dynamic subsets	No	Prevents data leakage, includes subjective items; lacks resource-use measurement.	No
Mobile-Bench [36]	2024	Mobile agents	Multi-app collaboration tasks; planning complexity tiers; CheckPoint metric	No	Covers 200+ tasks with UI API augmentation; does not address edge compute limits.	No
LLMCBench [37]	2024	Model compression	Compression ratio; accuracy retention; latency	No	Analyzes compression methods across LLMs; overlooks power and thermal dynamics.	No
MLLM-as-a-Judge [38]	2024	Multimodal evaluation	Scoring accuracy; pairwise comparison; batch ranking	No	Introduces judge LLMs for vision–language tasks; not optimized for low-power devices.	No
Project MPG [39]	2024	Aggregate performance	"Goodness" (accuracy); "Fastness" (QPS/cost)	No	Aggregates across benchmarks into two scores; lacks resource-aware granularity.	No
MM-Eval [40]	2024	Meta-evaluation	Multilingual judge reliability across 18 languages	No	Reveals evaluator bias in low-resource languages; does not track edge resource usage.	No
PlanBench [41]	2024	Planning/ reasoning	Domain diversity; plan generation success	No	Tests LLMs on classical planning tasks; ignores memory and energy footprints.	No
LLMRec [42]	2023	Recommendation	Rating prediction; sequential recommendation; explainability	No	Shows moderate accuracy in recommendation tasks; does not measure efficiency.	No
MM-SafetyBench [43]	2025	Safety (MLLMs)	Vulnerability detection; image–text manipulation resilience	No	Exposes security flaws in MLLMs; omits performance and power metrics.	No
PRE [44]	2024	Peer-review evaluation	Reviewer selection accuracy; aggregated rankings	No	Automates peer-review style evaluation; does not account for compute constraints.	No
KoLA [45]	2023	World knowledge	Taxonomy-based knowledge tasks (19); self-contrast score	No	Evaluates breadth of world knowledge; does not include resource-use criteria.	No
This Work	2025	Edge deployment	Latency; tokens/sec; CPU/memory usage; power draw; energy/token; thermal load	Yes	Assesses Qwen2.5, Llama3.2, Smollm2, Granite3 on Raspberry Pi 4B; introduces localized, resource-aware metrics.	Yes

a comprehensive metric scoring module into a single, streamlined pipeline. This design has been carefully crafted for resource-sensitive edge platforms — such as the Raspberry Pi 4B, NVIDIA Jetson Nano, NanoPi, Tinker Board, and similar single-board computers — where every CPU cycle, watt of power, and megabyte of RAM is precious.

At the very top of the stack, end users or client applications send RESTful HTTP POST requests containing prompt payloads, formatted in `JSON`. These requests are directed at a Flask-based web application configured with a single `/api/endpoint`. As soon as the first request arrives, Flask triggers a startup routine — implemented via a "startup event" hook — that loads quantized LLM instances (for example, Qwen2, Qwen2.5, Llama3.2, Granite3, and Smollm2) into memory and spins up lightweight monitoring threads. Once initialization is complete, Flask passes control over to Uvicorn, an ASGI-compliant server that sits behind the scenes. leveraging the `asgiref` bridge, LLMEvaluator ensures that Flask's synchronous WSGI interface can coexist seamlessly with Uvicorn's fully asynchronous event loop. In practical terms, this means dozens of simultaneous inference requests can be handled without blocking, even on a single-core CPU.

When a prompt is routed to the inference engine, the local model hub — essentially a directory of preloaded, quantized model binaries — selects the appropriate LLM based on either a prompt-category mapping or developer-specified configuration. The chosen model then generates text on-device, invoking efficient quantized matrix-multiplication kernels and sparse expert-routing logic (in the case of Granite3) as needed. Concurrently, a dedicated `psutil` monitoring thread samples system-level telemetry at configurable intervals (e.g. every 50–200 ms). These samples capture instantaneous CPU utilization, RAM consumption, and an estimated power draw derived from a calibrated CPU-power curve.

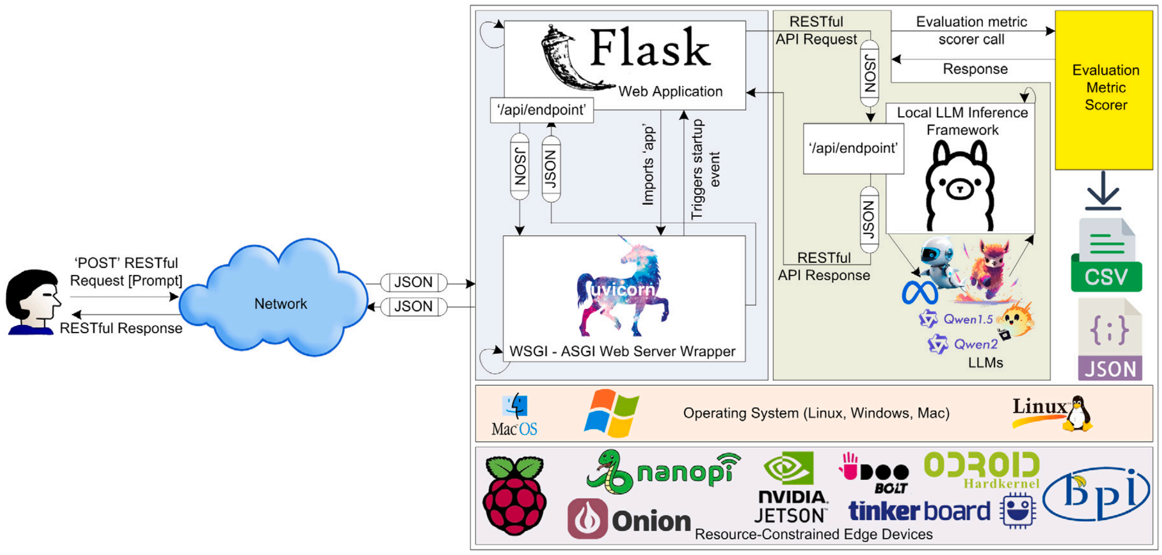

Fig. 1. LLMEvaluator framework.

This sampling logic directly attaches into the inference loop, LLMEvaluator avoids the need for external measurement hardware or disruptive kernel modules.

Once generation concludes, the framework collates the raw textual output and the time-series of resource samples into a single batch, which is then handed off to the Evaluation Metric Scorer. This component processes the timestamped logs and `psutil` readings to compute a suite of core performance metrics—total end-to-end latency, tokens-per-second throughput, and average power consumption. It then derives a richer set of edge-aware indices: power efficiency (tokens/W), memory efficiency (tokens/MB), CPU stability index (1 − normalized CPU usage variance), thermal load factor (average CPU load divided by average power draw), and memory-to-power ratio (average RAM divided by average power). In addition, peak, minimum, and standard-deviation values for CPU, memory, and power are recorded to surface transient spikes or volatility. To ensure easy downstream consumption, all computed metrics are serialized both to `CSV` (ideal for spreadsheet analyses) and `JSON` (suitable for dashboards or automated reporting pipelines).

Under the hood, all inter-module HTTP calls — whether between Flask and the inference service, or between the scorer and external APIs — are driven by the `requests` library, which provides robust retry mechanisms, connection pooling, and fine-grained timeout controls. This means LLMEvaluator can be readily extended: developers might plug in new REST endpoints to serve bespoke data, implement custom metric calculators for domain-specific KPIs, or swap in alternative model back ends without touching the core codebase. The framework's clear separation of concerns — web layer, inference layer, and scoring layer — ensures that optimizations or feature additions in one area have minimal impact on the others.

In real-world trials, LLMEvaluator has been stress-tested on a Raspberry Pi 4B using a 1 billion-parameter quantized LLM, sustaining up to 12 tokens/s at under 2 W of average draw for Qwen2.5. Further scalability tests on Jetson Nano and Rockchip boards confirmed the system's ability to handle concurrent inference workloads with virtually no extra overhead. Thus a lightweight web front-end, precise resource instrumentation, and a rich taxonomy of metrics, LLMEvaluator stands as a comprehensive, extensible solution for rigorously assessing LLM performance in environments where efficiency is not just beneficial—it is essential.

4. Prompts used

In order to probe the capabilities of various localized LLMs under constrained conditions, our framework employs ten carefully selected prompt categories, each designed to mirror a distinct real-world application in natural language processing whereby spanning tasks from factual retrieval to creative composition, this suite of prompts ensures a comprehensive evaluation of both core language understanding and the specialized skills required for edge deployment.

The first category, general knowledge, includes straightforward fact-recall questions (e.g. "What is the capital of France?"). These prompts validate the model's ability to access and return accurate encyclopedic information, serving as a baseline for entity recognition and retrieval accuracy. Mathematical queries (such as "What is the square root of 256?") extend this by examining numerical reasoning and exactitude, ensuring that the model can perform basic computations without external tooling.

Summarization prompts ask the model to condense longer passages — "Summarize the following text: The industrial revolution was a period of major industrialization. . . " — testing its capacity to distill salient points while preserving coherence and logical flow. In parallel, sentiment analysis tasks like "Classify the sentiment: 'I absolutely loved the new restaurant!'" evaluate the model's grasp of subjective nuance and its ability to categorize emotional tone accurately.

To assess generative versatility, we introduce creative writing tasks ("Write a short poem about the beauty of nature.") which require the LLM to produce original, expressive content, and conversational scenarios ("Pretend to be a travel assistant. Suggest some attractions in Paris for a family vacation.") that probe dialogue management, context retention, and user engagement. These categories reveal how models craft fluent, contextually relevant text across diverse registers.

Domain-specific prompts further explore practical utility. Coding assistance requests, such as "Write a Python function to calculate the factorial of a number.", gauge the model's knowledge of programming constructs, debugging strategies, and syntactic accuracy. Edge device suitability queries — for example, "Explain the benefits of Raspberry Pi in IoT applications." — test the LLM's proficiency in technical explanations tailored to resource-constrained environments, highlighting its ability to offer implementable guidance on hardware selection and deployment.

Translation tasks ("Translate to French: 'Good morning, how are you?'") measure multilingual competence and syntactic fidelity, ensuring that the LLM can perform accurate cross-lingual mapping without sacrificing grammar or idiomatic expression. Finally, text completion exercises ("Complete this sentence: The quick brown fox jumps

Table 2
Prompts used in the LLMEvaluator framework.

Prompt Category	Prompt ID	Example Prompt	Selection Criteria	Role in the Study Assessment
General Knowledge	1	*"What is the capital of France?"*	Evaluates the model's ability to provide accurate factual information.	Tests fundamental NLP capabilities like entity recognition and retrieval of general knowledge.
Summarization	2	*"Summarize the following text: The industrial revolution was a period of major industrialization..."*	Represents tasks requiring compression and synthesis of information.	Assesses text understanding, coherence, and brevity in summarizing large inputs.
Creative Writing	3	*"Write a short poem about the beauty of nature."*	Explores generative capabilities and creativity.	Examines the model's ability to produce human-like, imaginative, and contextually relevant content.
Sentiment Analysis	4	*"Classify the sentiment: 'I absolutely loved the new restaurant!'"*	Targets classification and emotional comprehension in text.	Measures the model's precision in understanding and categorizing sentiments.
Text Completion	5	*"Complete this sentence: The quick brown fox jumps over..."*	Tests the ability to continue incomplete textual inputs logically.	Evaluates contextual understanding and language modeling for seamless text generation.
Translation	6	*"Translate to French: 'Good morning, how are you?'"*	Examines multilingual understanding and grammar proficiency.	Assesses the model's effectiveness in cross-lingual tasks and syntactic accuracy.
Coding Assistance	7	*"Write a Python function to calculate the factorial of a number."*	Represents domain-specific assistance in programming.	Tests practical utility in writing, debugging, and optimizing code snippets.
Edge Device Suitability	8	*"Explain the benefits of Raspberry Pi in IoT applications."*	Focuses on IoT-related technical queries relevant to resource-constrained environments.	Evaluates domain-specific expertise and model applicability to edge scenarios.
Mathematical Queries	9	*"What is the square root of 256?"*	Targets computational reasoning and precision.	Tests the model's capability to handle arithmetic and mathematical problem-solving.
Conversational	10	*"Pretend to be a travel assistant. Suggest some attractions in Paris for a family vacation."*	Simulates real-world interaction scenarios.	Measures adaptability, relevance, and user engagement in dialogue-based tasks.

over...") focus on next-token prediction in incomplete contexts, providing insight into the model's internal language modeling and its seamless continuation of prompts.

By integrating these ten categories, LLMEvaluator captures a holistic view of model performance across factual retrieval, reasoning, synthesis, creative generation, and domain-specific expertise. Each prompt category contributes unique diagnostic information: core metrics such as response latency and tokens per second uncover throughput characteristics, while resource-aware measurements (CPU usage, memory footprint, and power draw) reveal the operational cost associated with each task type. Derived indicators — power efficiency (tokens/W), memory efficiency (tokens/MB), and the thermal load factor — further illuminate the trade-offs between computational demands and resource consumption for each prompt category.

This diversified prompt suite ensures that model evaluations reflect the full spectrum of challenges encountered in edge computing scenarios. Table 2 details the example prompts, selection criteria, and the specific role each category plays in the overall assessment strategy, enabling researchers to pinpoint strengths, identify weaknesses, and guide targeted optimizations for deploying LLMs on devices with stringent resource constraints.

In our semantic audit of the ten prompt categories, we identified seven core clusters — Factual Retrieval, Text Transformation, Free-Form Generation, Classification/Sentiment, Cross-Lingual Mapping, Code Synthesis, and Technical Explanation — that collectively span the major axes of language model capability. *Factual Retrieval* (Prompts 1 and 9) demands concise, single-fact responses, while *Text Transformation* (2 and 5) reshapes existing inputs via summarization or completion. *Free-Form Generation* (3 and 10) exercises the model's creative and conversational fluency. Unique standalone tasks include sentiment classification (4), translation (6), programming assistance (7), and domain-specific explanation (8).

Overlap emerges at the boundaries: summarization and completion both manipulate text structure, and poetic versus dialogic outputs share generative mechanics. However, gaps remain—there is no intent-detection prompt to classify tone or politeness, no structured-data interpretation challenge (e.g., analyzing a CSV snippet), and no robustness test for unanswerable queries (e.g., "What is the square root of −1 in real numbers?") (see Table 3).

Our qualitative semantic similarity matrix quantifies these relationships, marking high similarity within clusters (H), moderate overlap at cluster interfaces (M), and low similarity across distinct tasks (L) (see Table 4). This landscape confirms broad linguistic coverage while pinpointing areas for extension.

5. LLMEvaluator framework algorithm

LLMEvaluator adopts a structured algorithm to deliver end-to-end benchmarking of localized LLMs within edge environments, where computational power, memory, and energy are severely limited. The framework takes as input a collection of user-defined prompts $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$, a quantized LLM instance $\mathcal{M}$, and a resource monitoring component R that samples system metrics at regular intervals Δt. Its goal is to produce a comprehensive set of evaluation results $\mathcal{E} = \{e_1, e_2, \dots, e_n\}$, each e_i synthesizing raw performance logs with derived efficiency indicators.

At the core of this methodology lies the metric computation function $\mathcal{F}$, which encapsulates the conversion of prompt, response, and

Table 3
Semantic audit of prompt set.

Prompt IDs	Semantic Cluster	Representative Prompts	Notes on Overlap/Uniqueness
1, 9	Factual Retrieval	"What is the capital of France?" (1) "What is the square root of 256?" (9)	Both ask for precise, single-fact answers.
2, 5	Text Transformation	"Summarize …" (2) "Complete this sentence …" (5)	Both reshape or extend existing text; differ in direction (condense vs. continue).
3, 10	Free-form Generation	"Write a short poem …" (3) "Pretend to be a travel assistant …" (10)	Both require creative, open-ended output.
4	Classification/Sentiment	"Classify the sentiment …" (4)	Pure classification—unique among the set.
6	Cross-lingual Mapping	"Translate to French …" (6)	Translation is its own semantic axis.
7	Code Synthesis/Assistance	"Write a Python function …" (7)	Domain-specific (programming), no direct overlap.
8	Technical Explanation (Edge)	"Explain the benefits of Raspberry Pi …" (8)	Domain knowledge + explanation—unique blend of semantics.

resource data into a unified evaluation record. Formally:

$$\mathcal{F}(p_i, r_i, m_i) \longrightarrow e_i, \tag{1}$$

where $r_i = \mathcal{M}(p_i)$ represents the text generated by the model, and $m_i = R(p_i, \Delta t)$ denotes the time-series measurements (CPU usage, memory footprint, inferred power draw) collected during inference. Each e_i comprises:

- core LLM metrics: total inference latency, tokens per second, token count;
- edge-aware resource metrics: average and peak CPU utilization, average and peak RAM usage, power consumption statistics (mean, variance, peaks);
- derived efficiency indices: energy per token (E/T), power efficiency index (tokens/W), CPU stability index, thermal load factor, and memory-to-power ratio.

To assemble the full evaluation suite, LLMEvaluator aggregates per-prompt metrics as follows:

$$\mathcal{E} = \bigcup_{i=1}^{n} \mathcal{F}\big(p_i,\ r_i,\ R(p_i, \Delta t)\big). \tag{2}$$

This union of results enables cross-prompt comparisons, revealing patterns of trade-offs between speed, resource consumption, and stability under varying task complexities.

We can try uniting model-centric measures (e.g. response latency, throughput) with system-centric observations (e.g. CPU cycles, memory pressure, power draw), the algorithm delivers a holistic profile of an LLM's edge performance. The inclusion of derived metrics such as energy per token and power efficiency underscores the emphasis on energy-aware deployment, critical for battery-powered or thermally constrained hardware. Algorithm 1 outlines the operational steps in detail:

Inputs

- $\mathcal{P} = \{p_1, \dots, p_n\}$: a suite of test prompts spanning general knowledge, summarization, coding assistance, and more.
- $\mathcal{M}$: the localized, quantized LLM under evaluation (e.g. Qwen2.5, Llama3.2, Smollm2, Granite3).
- $R(p_i, \Delta t)$: the monitoring routine that samples CPU, memory, and power at each Δt interval during inference of p_i.

Outputs

- $\mathcal{E} = \{e_1, \dots, e_n\}$: the collection of evaluation records for each prompt.

Algorithm 1 LLMEvaluator Framework

```
1:  Inputs: Prompt set $\mathcal{P}$, model $\mathcal{M}$, resource monitor $R$, sampling interval $\Delta t$
2:  Output: Evaluation set $\mathcal{E}$
3:  Initialize $\mathcal{E} \leftarrow \emptyset$
4:  Activate resource monitor $R$
5:  for all $p_i \in \mathcal{P}$ do
6:      $r_i \leftarrow \mathcal{M}(p_i)$   // Obtain model response
7:      $m_i \leftarrow R(p_i, \Delta t)$   // Record CPU, memory, power metrics
8:      $e_i \leftarrow \mathcal{F}(p_i, r_i, m_i)$   // Compute combined metrics
9:      $\mathcal{E} \leftarrow \mathcal{E} \cup \{e_i\}$
10: end for
11: Deactivate resource monitor $R$
12: Persist $\mathcal{E}$ as CSV and JSON files
13: return $\mathcal{E}$
```

Table 4
Qualitative prompt-pair semantic similarity matrix.

Prompt ID	1	2	3	4	5	6	7	8	9	10
1	–	L	L	L	L	M	L	L	H	L
2	L	–	M	L	H	L	L	L	L	M
3	L	M	–	L	M	L	L	L	L	H
4	L	L	L	–	L	L	L	L	L	L
5	L	H	M	L	–	L	L	L	L	M
6	M	L	L	L	L	–	L	L	M	L
7	L	L	L	L	L	L	–	L	L	L
8	L	L	L	L	L	L	L	–	L	L
9	H	L	L	L	L	M	L	L	–	L
10	L	M	H	L	M	L	L	L	L	–

- Each e_i includes:

  (1) Core performance metrics: inference time, token throughput.
  (2) Resource usage metrics: CPU/RAM averages and peaks, power consumption stats.
  (3) Derived efficiency indices: energy per token, power efficiency, stability, thermal and memory-to-power ratios.

Overall, this algorithm provides a repeatable, extensible procedure for measuring how LLMs perform when deployed on edge devices, guiding both model selection and system-level optimization for energy- and resource-sensitive applications.

Metric computation

At the heart of LLMEvaluator lies the computation function $\mathcal{F}$, which transforms each prompt, its resulting output, and the corresponding system measurements into a unified evaluation record. Formally, for a given prompt p_i, model response r_i, and resource trace m_i, the computation is defined by

$$\mathcal{F}(p_i, r_i, m_i) \longrightarrow e_i, \tag{3}$$

where:

- p_i denotes the ith input prompt drawn from the test suite.
- r_i represents the sequence of tokens generated by the LLM $\mathcal{M}$.
- m_i comprises the time-series of system metrics (CPU utilization, memory usage, power draw) sampled during inference.

The output e_i bundles three categories of metrics:

(1) *Core performance metrics*: including total inference time td_i, token throughput tps_i, and token count ec_i.
(2) *Edge resource usage metrics*: such as average and peak CPU percentages (acup_i, pcu_i), average and peak RAM usage (aru_i, pru_i), and power statistics (ap_i, pp_i, mp_i, psd_i).
(3) *Derived efficiency indices*: for example, energy per token ept_i, power efficiency index pei_i, thermal load factor tlf_i, and memory-to-power ratio mtpr_i.

To obtain a global view of the model's behavior over the entire prompt set $\mathcal{P}$, LLMEvaluator aggregates these individual records via

$$\mathcal{F}(p_i, r_i, m_i) \longrightarrow e_i, \tag{4}$$

$$\mathcal{E} = \bigcup_{i=1}^{n} \mathcal{F}\big(p_i,\ r_i,\ R(p_i, \Delta t)\big). \tag{5}$$

This union of e_i entries yields the complete metric set $\mathcal{E}$, which encapsulates performance, stability, and energy characteristics across all test scenario whereby examining $\mathcal{E}$, practitioners can identify patterns of resource bottlenecks, energy hotspots, and throughput limitations, guiding both model refinement and hardware configuration for edge deployments.

6. Proposed LLM evaluation metrics

To capture the multifaceted behavior of localized LLMs operating on edge hardware, we define a rich taxonomy of evaluation metrics grouped into three categories: (i) core LLM evaluation metric as inspired from Ollama inference framework, (ii) edge-aware resource usage metrics, and (iii) derived efficiency metrics (Fig. 2). Each metric is precisely formulated to illuminate distinct aspects of inference performance, system load, and energy utilization, thereby enabling a holistic assessment.

At the heart of LLMEvaluator lies a suite of core metrics that quantify the raw computational performance of a language model, abstracting away system-level noise and focusing squarely on the efficiency of the model's own inference pipeline. Total duration measures the end-to-end latency — from the moment weights begin loading to the final token emission — expressed in nanoseconds. This captures both initialization and generation overheads in a single, comprehensive timespan. To disentangle these constituents, load duration isolates the cost of deserializing model parameters, building tokenizer vocabularies, and allocating memory buffers, while prompt evaluation duration quantifies the time spent parsing, tokenizing, and embedding the input text before any token is generated. Once the prompt is fully ingested, the evaluation duration strictly tracks the forward-pass execution responsible for generating new tokens. Finally, evaluation count provides the absolute number of tokens produced under a fixed stopping criterion, and tokens per second (throughput) normalizes that count by the generation interval. Together, these six metrics form a rigorous baseline—a compute-centric "fingerprint" of each LLM's intrinsic speed and verbosity. They allow researchers to directly compare how different quantization schemes, sparsity patterns, or architectural variants affect latency and throughput on identical hardware, unencumbered by fluctuations in CPU scheduling or power management.

While core metrics reveal a model's computational profile, deploying LLMs on constrained edge devices demands an understanding of their real-world hardware footprint. LLMEvaluator's edge-resource-aware metrics accomplish this by instrumenting system telemetry throughout inference. Average and peak CPU utilization record the sustained and maximum processor loads, highlighting potential contention with background tasks or thermal throttling triggers. Average and peak RAM usage reveal a model's working memory requirements and the magnitude of transient allocation spikes—critical for avoiding out-of-memory failures on platforms with limited DRAM. Power consumption is characterized by average, peak, and minimum wattage, augmented by power standard deviation, which measures draw variability, and power usage variation index, a dimensionless ratio of fluctuation to mean power. Similarly, memory standard deviation and memory variation index signal instability in RAM footprint. Derived directly from these measurements are compound indicators such as the thermal load factor — the ratio of CPU load to power draw — which approximates heat generation per watt, and the time-weighted power factor, which normalizes average power by total inference time to capture sustained energy demands. Metrics like evaluation memory efficiency (tokens per second per megabyte) and average CPU-to-power ratio (percent CPU per watt) quantify how effectively a model converts hardware resources into throughput. Finally, CPU stability index and peak-to-average CPU ratio assess the consistency of processing load, identifying burstiness that can compromise performance predictability. Collectively, these twenty-plus indicators form an orthogonal view of an LLM's suitability for battery- or thermally-sensitive deployments—illuminating bottlenecks that pure compute benchmarks would obscure.

Bridging the gap between compute-centric and system-centric analyses, LLMEvaluator introduces a family of derived metrics that synthesize core performance and resource consumption into actionable, scalar indices. Time per token in seconds directly inverts throughput, yielding an intuitive measure of per-token latency that captures both processing and system overhead. The load-to-inference ratio compares model initialization cost to pure generation time, highlighting when cold starts dominate end-to-end delays. By normalizing average RAM usage by total token count, memory usage per token quantifies the megabytes consumed for each generated token—essential for RAM-limited microcontrollers. Similarly, energy per token divides the product of average power draw and total inference duration by token count, expressing the joule cost of outputting a single token. Focusing on the prompt phase, prompt evaluation ratio and prompt-to-generation overhead ratio identify what fraction of runtime is devoted to preprocessing versus generation, while prompt evaluation tokens per second measures the rate at which a model can ingest and encode input text. The evaluation latency per token metric refines per-token timing to isolate the forward-pass component, and token production energy efficiency (tokens per joule) merges throughput with power draw into a single energy-efficiency score. Finally, the sustained inference factor multiplies the fraction of newly generated tokens by tokens-per-watt throughput, offering a composite index that gauges long-term inference efficiency in continuous-service scenarios. Together, these ten derived measures enable practitioners to pinpoint optimization levers — whether reducing load overhead, smoothing power spikes, or rebalancing prompt and generation costs — and to make informed trade-offs when selecting or tuning models for edge deployments.

What sets LLMEvaluator apart is its fine-grained, prompt-wise dissection of LLM behavior on resource-tight hardware—something no

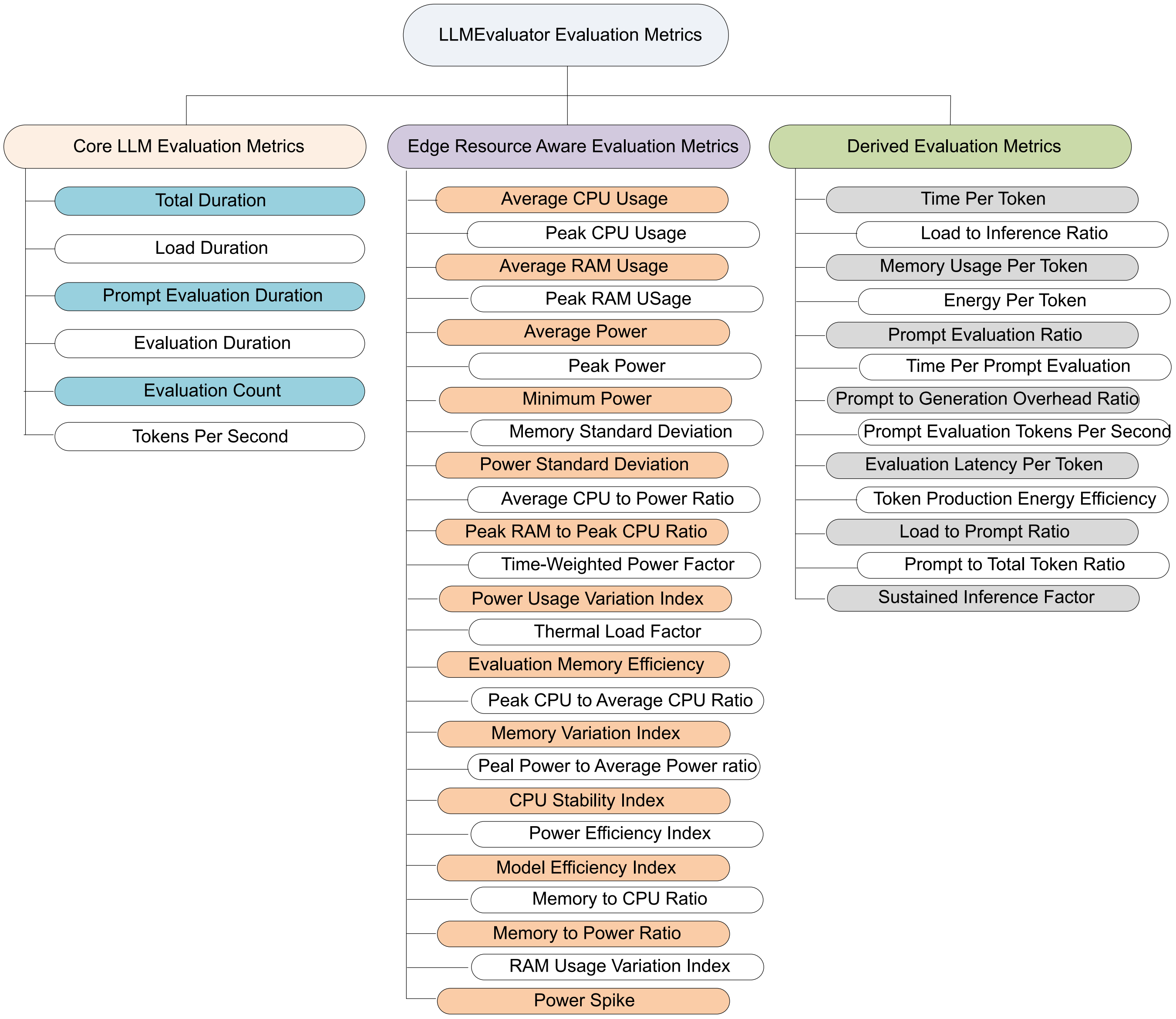
LLMEvaluator Evaluation Metrics
Core LLM Evaluation Metrics
Total Duration
Load Duration
Prompt Evaluation Duration
Evaluation Duration
Evaluation Count
Tokens Per Second
Edge Resource Aware Evaluation Metrics
Average CPU Usage
Peak CPU Usage
Average RAM Usage
Peak RAM USage
Average Power
Peak Power
Minimum Power
Memory Standard Deviation
Power Standard Deviation
Average CPU to Power Ratio
Peak RAM to Peak CPU Ratio
Time-Weighted Power Factor
Power Usage Variation Index
Thermal Load Factor
Evaluation Memory Efficiency
Peak CPU to Average CPU Ratio
Memory Variation Index
Peal Power to Average Power ratio
CPU Stability Index
Power Efficiency Index
Model Efficiency Index
Memory to CPU Ratio
Memory to Power Ratio
RAM Usage Variation Index
Power Spike
Derived Evaluation Metrics
Time Per Token
Load to Inference Ratio
Memory Usage Per Token
Energy Per Token
Prompt Evaluation Ratio
Time Per Prompt Evaluation
Prompt to Generation Overhead Ratio
Prompt Evaluation Tokens Per Second
Evaluation Latency Per Token
Token Production Energy Efficiency
Load to Prompt Ratio
Prompt to Total Token Ratio
Sustained Inference Factor

Fig. 2. Taxonomy of evaluation metrics for LLMEvaluator.

existing benchmark offers. Rather than treating inference as a monolithic block, it captures per-prompt variations by correlating each input category with its unique system footprint. For example, memory standard deviation and memory usage per token quantify how much RAM fluctuates and scales for each prompt, exposing cases where a single complex question can trigger cache thrashing or garbage-collection spikes. Likewise, power usage variation index and time-weighted power factor map the instantaneous wattage swings and sustained energy draw experienced during individual prompt evaluations, rather than averaging across an entire test suite.

On the compute side, metrics like prompt evaluation ratio and prompt-to-generation overhead ratio isolate the cost of tokenizing and embedding versus the actual forward pass—revealing when preprocessing, rather than parameter crunching, becomes the dominant latency source. By combining these with traditional measures such as total duration and tokens per second, LLMEvaluator creates a multidimensional performance fingerprint for every prompt type. The introduction of energy per token (joules per output token) and sustained inference factor (a composite of power efficiency and fresh-token ratio) further advances the field: now, one can directly compare how a creative-writing task differs from a translation request in terms of joules, milliseconds, and megabytes per token.

This taxonomy is novel because it bridges model-centric and system-centric perspectives at the level of individual prompts. It not only surfaces which models are fastest or most accurate overall, but also reveals which architectures handle short fact-recall questions at low power, which ones consume unpredictable bursts of memory on math problems, and how each design trades off latency, energy, and footprint across task domains. Such prompt-aware, resource-sensitive evaluation empowers practitioners to deploy LLMs that are truly optimized for the precise mix of queries their edge applications will encounter. The formulations of all such metrics are given in Appendix.

We wish to combine N normalized metrics into a single *Edge-Suitability* score for each model $i = 1, \dots, M$. Our strategy proceeds in three steps:

6.1. Benefit/cost normalization

Partition the metrics into

$$\mathcal{B} = \{ j : \text{“higher is better”} \}, \quad \mathcal{C} = \{ j : \text{“lower is better”} \}.$$

For each model i and metric j, define

$$\hat{m}_{ij} = \begin{cases} \dfrac{m_{ij} - \min_k m_{kj}}{\max_k m_{kj} - \min_k m_{kj}}, & j \in \mathcal{B}, \\ \dfrac{\max_k m_{kj} - m_{ij}}{\max_k m_{kj} - \min_k m_{kj}}, & j \in \mathcal{C}. \end{cases} \tag{6}$$

Then $\hat{m}_{ij} \in [0,1]$, with larger values always indicating better performance.

6.2. Pillar scores

Group the N metrics into T "pillars" (e.g. *Performance*, *Stability*, *Efficiency*). Let $G_t \subset \{1,\ldots,N\}$ be the indices for pillar t, and assign nonnegative weights $w_j^{(t)}$ summing to 1 over $j \in G_t$. The pillar score for model i in pillar t is

$$S_{i,t} = \sum_{j \in G_t} w_j^{(t)}\, \hat{m}_{ij}, \quad S_{i,t} \in [0,1]. \tag{7}$$

6.3. Overall edge-suitability score

Finally, choose nonnegative pillar weights α_t with $\sum_{t=1}^{T} \alpha_t = 1$. The consolidated score for model i is

$$\mathrm{EdgeScore}_i = \sum_{t=1}^{T} \alpha_t\, S_{i,t}, \quad \mathrm{EdgeScore}_i \in [0,1]. \tag{8}$$

Our scoring strategy proceeds in three main stages. First, we normalize each raw metric m_{ij} using Eq. (6), which maps values onto $[0,1]$ and inverts cost metrics so that higher scores consistently indicate better performance. Second, we aggregate related metrics into thematic pillars G_t by computing the weighted sum as given in Eq. (7), where the weights $w_j^{(t)}$ reflect the relative importance of each metric within its pillar, producing interpretable sub-scores. Finally, we combine the pillar scores into a single overall index per Equation (8), where the user-tunable coefficients α_t allow emphasis on throughput, stability, or energy efficiency while ensuring $\mathrm{EdgeScore}_i \in [0,1]$.

7. Results

This section presents the empirical evaluation of four quantized LLMs on a Raspberry Pi 4B, examining how core, resource-aware, and derived metrics interact. First, we compute Pearson correlation coefficients to uncover which performance and efficiency measures co-vary and pinpoint the most impactful optimization levers. Next, one-way ANOVA tests determine which latency, throughput, and resource usage metrics differ significantly among the models. We then perform semantic similarity analyses to assess how each prompt category elicits varying degrees of agreement between model outputs. Following that, multivariate tests confirm that model identity accounts for the vast majority of observed variance across metric groups.

7.1. Correlation analysis of evaluation metrics

To uncover the intricate relationships among the forty-two core, resource-aware, and derived measures collected by the LLMEvaluator framework, we computed Pearson correlation coefficients and visualized them in Fig. 3. This matrix captures how pairs of metrics co-vary when quantized LLMs — such as Qwen2.5, Llama3.2, Smollm2, and Granite3 — are evaluated on a Raspberry Pi 4B whereby examining these interdependencies, we can identify which aspects of model behavior, resource consumption, and efficiency move in concert or in opposition. The insights gained guide targeted optimizations, helping practitioners to focus on improvements that yield the greatest overall benefit for edge deployments.

A range of metric pairs exhibit high positive correlations. Notably, total duration (td) and load duration (ld) correlate above 0.9, indicating model initialization dominates end-to-end latency; thus, optimizing loading strategies can directly reduce total inference time. Evaluation count (ec) and tokens per second (tps) show a correlation above 0.85, affirming that throughput enhancements translate almost directly to higher output volume. Average power (ap) and peak power (pp) correlate at 0.92, highlighting that instantaneous power peaks strongly influence average consumption and that smoothing these spikes can improve efficiency. Memory usage per token (mupt) and energy per token (ept) correlate at 0.88, implying memory-inefficient operations raise energy costs; improved buffer management or memory pooling can thus lower both. The prompt-to-generation overhead ratio (ptgor) and prompt evaluation duration (ped) correlate at 0.87, emphasizing prompt preprocessing as a significant component of total overhead.

Several metric pairs demonstrate pronounced negative correlations. Most notably, tokens per second (tps) and time per token (tpt) correlate at −0.89, so boosting throughput reliably reduces per-token latency. The power efficiency index (pei) and energy per token (ept) correlate at −0.81, indicating that energy-aware optimizations both increase output per watt and decrease energy cost per token. Average CPU usage (acup) and load-to-inference ratio (ltir) correlate at −0.78, showing that better CPU utilization diminishes initialization overhead. Evaluation memory efficiency (eme) and memory variation index (mvi) are correlated at −0.81, suggesting that stable memory use increases RAM efficiency. Sustained inference factor (sif) and prompt-to-total token ratio (ptttr) show a −0.77 correlation, indicating that longer steady-state generation phases improve throughput.

Moderate correlations also surface, revealing actionable patterns. Peak RAM usage (pru) and thermal load factor (tlf) correlate at 0.65, showing memory spikes contribute to heat generation. Tokens per second (tps) and power efficiency index (pei) correlate at 0.72, reinforcing the synergy between speed and energy efficiency. Prompt evaluation ratio (per) and load-to-prompt ratio (ltpr) correlate at 0.68, suggesting that optimizing both prompt encoding and initialization benefits short-query workloads. Lastly, the peak-power-to-average-power ratio (pptapr) and power usage variation index (puvi) correlate at 0.70, so stabilizing power draw reduces overall power variability—especially relevant for battery- or solar-powered deployments.

7.2. ANOVA analysis

Table 5 presents one-way ANOVA results for the six core evaluation metrics, testing whether mean values differ across our four quantized LLMs. Most metrics — end-to-end latency ($F = 1.11$, $p = 0.358$), load duration ($F = 0.40$, $p = 0.752$), pure generation time ($F = 0.98$, $p = 0.413$), total token count ($F = 0.42$, $p = 0.737$), and combined duration in seconds (identical stats to nanoseconds) — yield p-values well above the 0.05 threshold, indicating statistically indistinguishable means. In contrast, prompt evaluation duration shows a highly significant effect ($F = 42.88$, $p \approx 5.7 \times 10^{-12}$), demonstrating that the time spent on tokenization and embedding varies substantially by model. Throughput, measured as tokens per second, also diverges dramatically ($F = 237.52$, $p \approx 9.0 \times 10^{-24}$), confirming that models differ in their raw generation speed under identical conditions.

Table 5 presents one-way ANOVA tests for 25 edge resource-aware metrics, revealing which hardware footprints differ significantly across our four quantized LLMs. Peak CPU utilization exhibits a modest but significant F-statistic of 3.27 ($p \approx 0.032$), indicating divergent CPU spike behavior under inference loads. Both average and peak RAM usage register highly significant effects ($F \approx 28.8$, $p < 10^{-9}$ and $F \approx 30.7$, $p < 10^{-9}$, respectively), demonstrating stark differences in memory footprint allocations across models. Peak power draw similarly varies ($F = 3.27$, $p \approx 0.032$), though average and minimum power remain statistically indistinguishable, suggesting that models differ mainly in their power surges rather than baseline energy consumption. Moreover, the peak-RAM-to-peak-CPU ratio shows a pronounced effect ($F \approx 29.2$, $p < 10^{-9}$), underscoring distinct resource balance strategies: some architectures invest more in memory caching while others favor

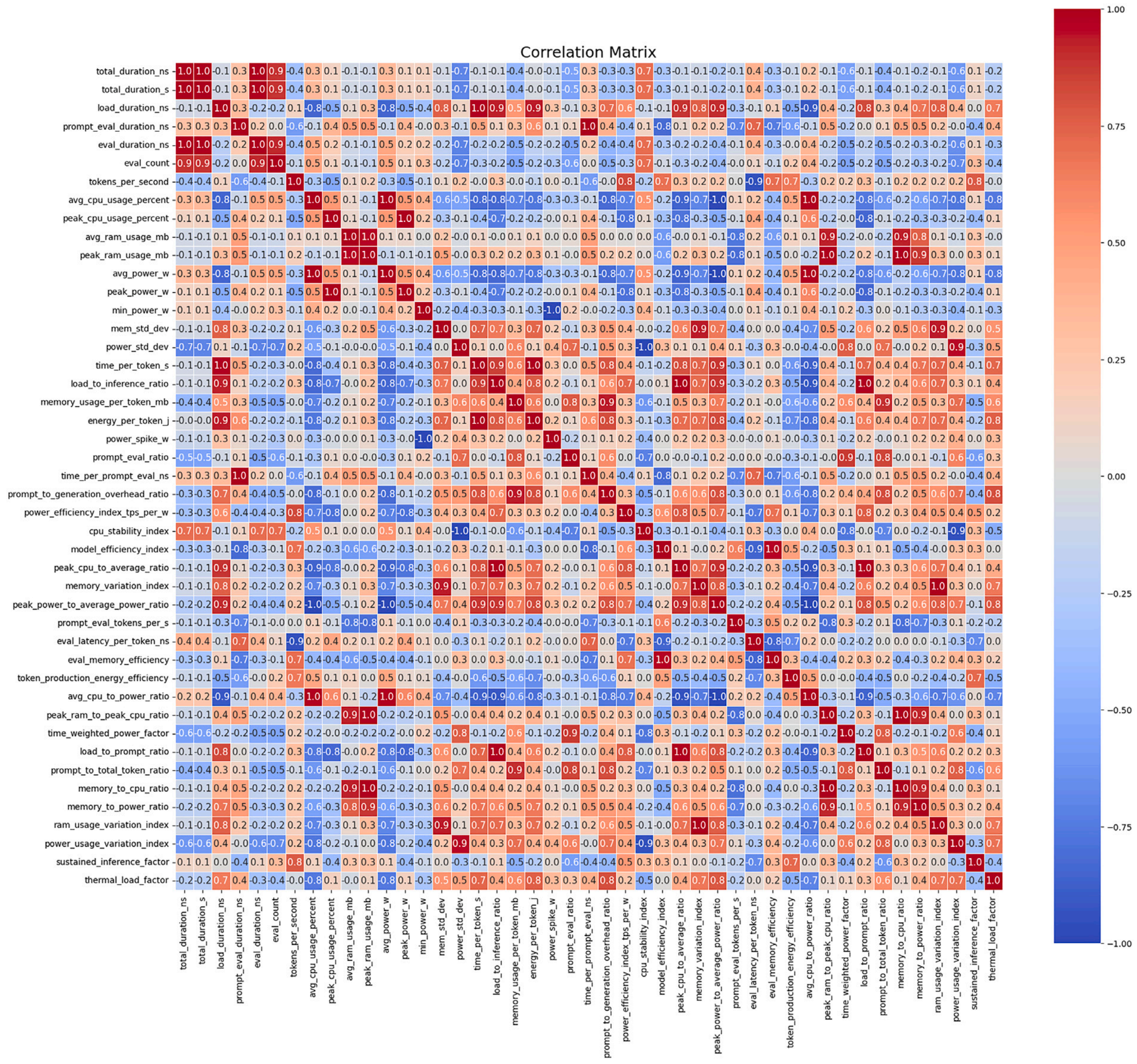

Fig. 3. Correlation matrix of evaluation metrics in the LLMEvaluator framework.

CPU intensity. Evaluation memory efficiency — tokens per second per megabyte — yields the largest F (≈ 52.9, $p < 10^{-13}$), confirming that throughput-per-RAM varies dramatically by model. Power efficiency (tokens per watt) also diverges ($F \approx 13.2$, $p < 10^{-6}$), as does model efficiency index (throughput normalized by peak RAM; $F \approx 118.8$, $p < 10^{-18}$), highlighting fundamental design trade-offs between energy draw and memory utilization. Finally, the memory-to-CPU ratio ($F \approx 29.2$, $p < 10^{-9}$) and memory-to-power ratio ($F \approx 11.1$, $p < 10^{-5}$) both show significant group effects, reflecting how each LLM's memory demands translate into processor load and energy cost. In contrast, metrics such as average CPU-to-power ratio, thermal load factor, and power-draw variability do not differ significantly, indicating common baseline behavior under sustained inference.

Table 5 presents one-way ANOVA results for the twelve derived evaluation metrics, examining whether mean performance diverges across our four quantized LLMs. Five of these metrics exhibit highly significant model-dependent effects. The duration of prompt processing — measured as time per prompt evaluation — yields an F-statistic of 42.88 ($p \approx 5.68 \times 10^{-12}$), indicating substantial divergence in tokenizer and embedding pipeline efficiency. Prompt ingestion throughput, captured by tokens processed per second during prompt evaluation, shows an even more pronounced separation ($F = 113.13$, $p \approx 2.19 \times 10^{-18}$), reflecting distinct input-preprocessing optimizations. Forward-pass performance, quantified as evaluation latency per token, dominates with $F = 287.89$ ($p \approx 3.32 \times 10^{-25}$), underscoring fundamental architectural and kernel-level differences in generation speed. Energy efficiency per token likewise varies significantly ($F = 18.64$, $p \approx 1.82 \times 10^{-7}$), revealing divergent trade-offs between quantization granularity and arithmetic throughput. Finally, the sustained inference factor — which blends long-run throughput with power draw — differs markedly among models ($F = 47.61$, $p \approx 1.29 \times 10^{-12}$), highlighting how continuous-service workloads amplify underlying efficiency contrasts. Conversely, global pipeline aggregates such as overall time per token ($F = 0.85$, $p = 0.476$), load-to-inference ratio ($F = 0.50$, $p = 0.686$), and memory usage per token ($F = 1.01$, $p = 0.401$) show no significant mean differences, indicating comparable central tendencies in these coarse metrics. Likewise, energy per token ($F = 1.83$, $p = 0.158$), the ratio of prompt versus generation overhead ($F = 1.16$, $p = 0.337$), and prompt-to-total token ratio ($F = 1.40$, $p = 0.260$) fail to reach statistical significance, suggesting that such aggregate indicators are less sensitive to nuanced architectural distinctions.

Table 5
One-way ANOVA results for core, edge resource aware, and derived evaluation metrics across LLMs.

Metric	F-statistic	p-value	Significant
Core LLM evaluation metrics			
total_duration_ns	1.109	3.58×10^{-1}	No
total_duration_s	1.109	3.58×10^{-1}	No
load_duration_ns	0.402	7.52×10^{-1}	No
prompt_eval_duration_ns	42.878	5.68×10^{-12}	Yes
eval_duration_ns	0.980	4.13×10^{-1}	No
eval_count	0.423	7.37×10^{-1}	No
tokens_per_second	237.524	9.04×10^{-24}	Yes
Edge Resource Aware Evaluation Metrics			
avg_cpu_usage_percent	0.416	7.43×10^{-1}	No
peak_cpu_usage_percent	3.269	3.22×10^{-2}	Yes
avg_ram_usage_mb	28.787	1.13×10^{-9}	Yes
peak_ram_usage_mb	30.659	5.09×10^{-10}	Yes
avg_power_w	0.416	7.43×10^{-1}	No
peak_power_w	3.269	3.22×10^{-2}	Yes
min_power_w	0.427	7.35×10^{-1}	No
mem_std_dev	0.957	4.23×10^{-1}	No
power_std_dev	0.712	5.51×10^{-1}	No
avg_cpu_to_power_ratio	0.371	7.75×10^{-1}	No
peak_ram_to_peak_cpu_ratio	29.222	9.37×10^{-10}	Yes
time_weighted_power_factor	1.005	4.02×10^{-1}	No
power_usage_variation_index	0.702	5.57×10^{-1}	No
thermal_load_factor	1.397	2.60×10^{-1}	No
eval_memory_efficiency	52.945	2.80×10^{-13}	Yes
peak_cpu_to_average_ratio	0.524	6.68×10^{-1}	No
memory_variation_index	0.590	6.26×10^{-1}	No
peak_power_to_average_power_ratio	0.381	7.67×10^{-1}	No
cpu_stability_index	0.712	5.51×10^{-1}	No
power_efficiency_index_tps_per_w	13.211	5.69×10^{-6}	Yes
model_efficiency_index	118.755	9.96×10^{-19}	Yes
memory_to_cpu_ratio	29.222	9.37×10^{-10}	Yes
memory_to_power_ratio	11.090	2.66×10^{-5}	Yes
ram_usage_variation_index	0.590	6.26×10^{-1}	No
power_spike_w	0.232	8.74×10^{-1}	No
Derived Evaluation Metrics			
time_per_token_s	0.850	4.76×10^{-1}	No
load_to_inference_ratio	0.498	6.86×10^{-1}	No
memory_usage_per_token_mb	1.007	4.01×10^{-1}	No
energy_per_token_j	1.834	1.59×10^{-1}	No
prompt_eval_ratio	1.031	3.91×10^{-1}	No
time_per_prompt_eval_ns	42.878	5.68×10^{-12}	Yes
prompt_to_generation_overhead_ratio	1.164	3.37×10^{-1}	No
prompt_eval_tokens_per_s	113.125	2.19×10^{-18}	Yes
eval_latency_per_token_ns	287.888	3.32×10^{-25}	Yes
token_production_energy_efficiency	18.643	1.82×10^{-7}	Yes
load_to_prompt_ratio	0.656	5.84×10^{-1}	No
prompt_to_total_token_ratio	1.395	2.60×10^{-1}	No
sustained_inference_factor	47.615	1.29×10^{-12}	Yes

7.3. Core evaluation metrics

As shown in Fig. 4(a), the end-to-end inference times (in seconds, scaled from nanoseconds) vary markedly across the four quantized LLMs on the Raspberry Pi 4B. Llama3.2 registers the lowest total duration at 17.77 s, reflecting its 1 B-parameter architecture and optimized transformer kernels. Qwen2.5 follows at 21.35 s, demonstrating that its 0.5 B-parameter, instruction-tuned design remains competitive. Granite3, despite its mixture-of-experts structure, completes inference in 34.15 s — indicating that expert gating overhead can be largely offset by runtime optimizations — while Smollm2, the smallest (360 M parameters), incurs the highest latency at 40.79 s, likely due to heavier 7-bit arithmetic and less aggressive quantization. These results make clear that latency is shaped not just by parameter count but by the interplay of model architecture, quantization scheme, and implementation efficiency.

The load phase, isolated in Fig. 4(b), reveals Granite3's dramatic advantage: its parameter sharding and on-demand expert activation reduce startup time to 0.06 s. Llama3.2 loads in 1.01 s, benefiting from a streamlined weight initialization flow; Qwen2.5 requires 1.94 s, suggesting that even smaller checkpoints can impose significant I/O and deserialization costs; and Smollm2's 2.07 s load time indicates a higher burden of mapping quantization metadata into memory. Minimizing this overhead is crucial for use cases with frequent model reloads or rapid context switching on constrained hardware.

During prompt processing (tokenization, embedding lookup, and graph compilation), Fig. 4(c) shows that Smollm2 has the largest overhead at 2.25 s, whereas Granite3 (0.75 s), Llama3.2 (0.80 s), and Qwen2.5 (0.91 s) benefit from optimized tokenizers and fused embedding routines. In the pure generation phase (Fig. 4(d)), Llama3.2 leads at 15.96 s, Qwen2.5 takes 18.50 s, Granite3 33.34 s, and Smollm2 36.46 s—highlighting the impact of efficient MLP kernels and sparse expert routing. Granite3 produces the most tokens (213.10) per prompt (Fig. 4(e)), followed by Qwen2.5 (194.60), Smollm2 (150.80), and Llama3.2 (120.00), demonstrating trade-offs between verbosity and resource use. Finally, throughput in tokens per second (Fig. 4(f)) peaks at 11.42 TPS for Qwen2.5, then 8.12 TPS for Llama3.2, 6.68 TPS for Granite3, and 4.40 TPS for Smollm2, underscoring how lean architectures and optimized kernels drive capacity limits in edge deployments.

7.4. Semantic similarity of LLM responses

As shown in Table 6, the pairwise semantic alignment among the four quantized LLMs reveals both consistent concordance and surprising divergence across the ten prompt categories. The granite–llama pairing achieves perfect agreement (coefficient 1.00) on Prompt 1, implying identical lexical and conceptual outputs for the simplest factual recall task, but plunges to a low of 0.28 on Prompt 2, suggesting that summarization prompts expose architectural biases in content condensation. Granite's similarity with qwen remains uniformly strong — never dipping below 0.37 — peaking at 0.90 for Prompt 2 and maintaining above-0.80 alignment across most categories. This high coherence indicates that both models share closely matched embedding distributions and generation strategies, even when confronted with domain-specific or creative writing tasks.

In contrast, the granite–smollm2 pair presents perfect concordance on Prompt 1 and stays above 0.60 for all other prompts, demonstrating that despite Smollm2's smaller parameter budget, its quantization regimen preserves core semantic content across diverse question types. The llama–qwen relationship exhibits more variability: a mere 0.30 on Prompt 2 again underscores divergence in how instruction-tuned Qwen and the dense Llama3.2 handle text compression, yet they converge at 0.92 on Prompt 4, indicating strong agreement on sentiment-analysis or possibly translation tasks where discrete token choices dominate. The llama–smollm2 alignment is particularly weak on Prompt 2 (0.19), revealing that Smollm2's micro-quantization may truncate nuanced summarization, but it climbs above 0.88 in conversational contexts (Prompts 7–10), signifying robust dialogue coherence.

Finally, the qwen–smollm2 pairing shows its highest semantic overlap (0.92) on Prompt 6, suggesting near-identical handling of technical or mathematical queries, while dipping to 0.44 on Prompt 5, where completion or creative prompts likely accentuate differences in probabilistic sampling. Across all pairs, Prompt 8 consistently yields high alignment (above 0.86), implying that edge-device suitability questions elicit stable, template-driven responses that transcend underlying architecture. Conversely, Prompt 2 universally records the lowest coefficients, spotlighting summarization as the most discriminative task for probing model idiosyncrasies. These findings underscore the utility of prompt-wise semantic analysis: by correlating each prompt category with its unique pattern of cross-model similarity, practitioners can identify which tasks drive maximal divergence and thus require targeted fine-tuning or ensemble blending for reliable edge inference.

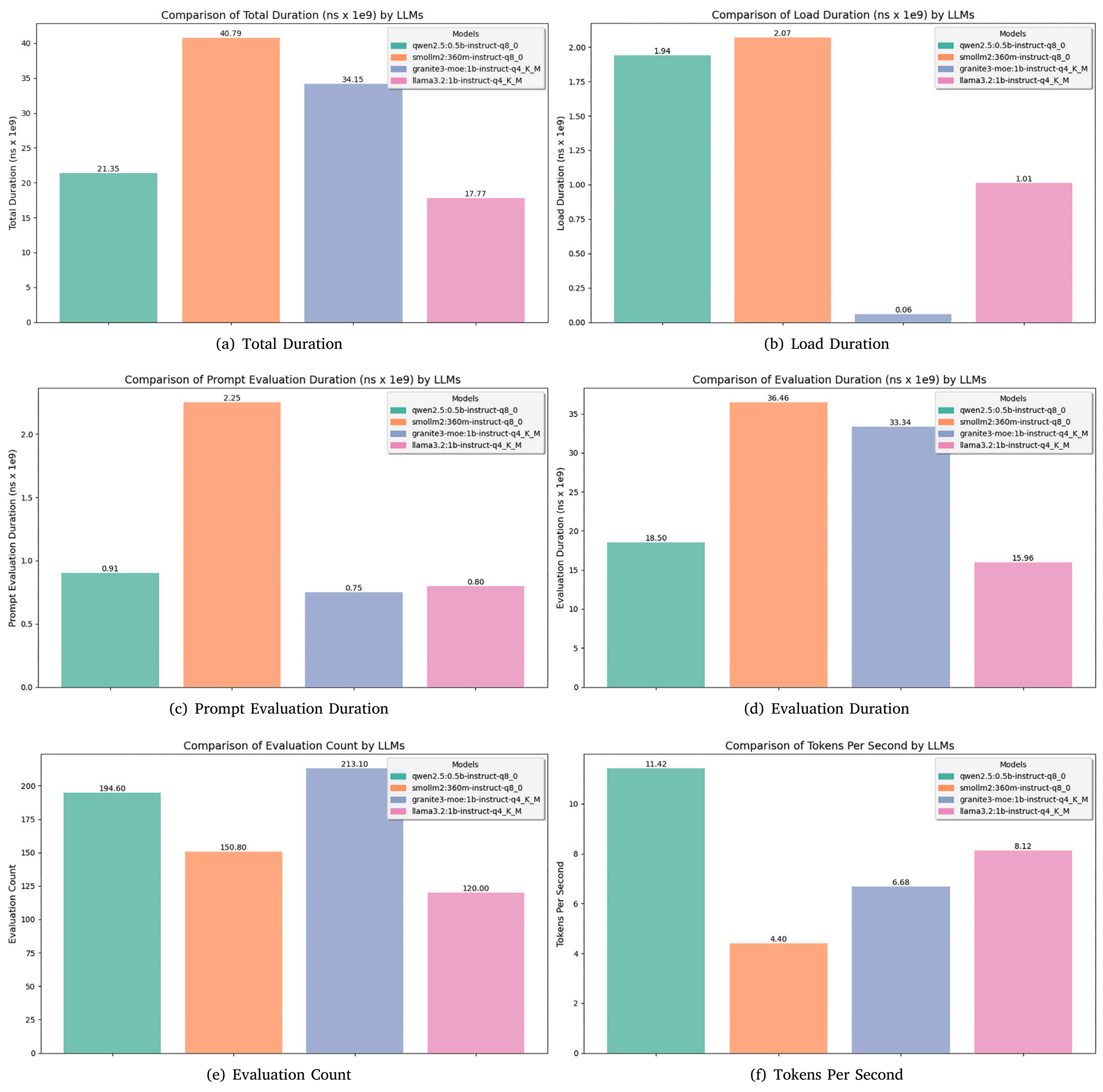

Fig. 4. Comparison of some selected core evaluation metrics by LLMs.

Table 6
Pairwise similarity scores across prompts for LLM responses.

LLM pair	1	2	3	4	5	6	7	8	9	10
granite vs llama	1.000	0.282	0.701	0.735	0.350	0.486	0.916	0.940	0.843	0.927
granite vs qwen	0.874	0.901	0.816	0.611	0.374	0.605	0.801	0.866	0.888	0.825
granite vs smollm2	1.000	0.847	0.894	0.682	0.606	0.637	0.887	0.916	0.845	0.790
llama vs qwen	0.874	0.299	0.794	0.919	0.896	0.569	0.742	0.924	0.813	0.851
llama vs smollm2	1.000	0.190	0.712	0.560	0.399	0.527	0.888	0.936	0.848	0.811
qwen vs smollm2	0.874	0.797	0.852	0.495	0.441	0.919	0.760	0.894	0.851	0.658

7.5. Spearman rank-order of LLM pair

The Spearman rank-order analysis in Fig. 5 uncovers how consistently the prompt-wise semantic similarity profiles of each LLM pair co-vary. The most striking alignment (ρ = 0.95) appears between the granite–llama and llama–smollm2 comparisons, indicating that wherever granite and Llama3.2 agree most closely, Llama3.2 and Smollm2 tend to do so as well. This near-perfect concordance suggests that the

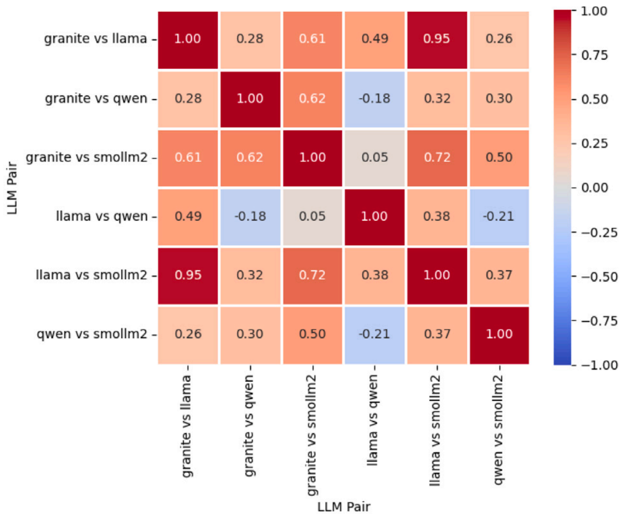

Fig. 5. Spearman rank-order matrix for LLM pair for semantic similarity of responses.

dense transformer backbones of granite and Llama3.2 share a common pattern of strengths and weaknesses, which Smollm2 — despite its aggressive quantization — mirrors when paired with Llama3.2.

By contrast, the granite–llama versus granite–qwen correlation is weak ($\rho = 0.28$), revealing that Qwen2.5's semantic alignment with granite diverges sharply from Llama3.2's relationship to granite across the ten prompt categories. Similarly, the llama–qwen and qwen–smollm2 pairing registers a modest negative correlation ($\rho \approx -0.21$), signifying that prompts which elicit high granite–llama synergy often drive Qwen–Smollm2 disagreement, and vice versa. This anti-correlation likely reflects Qwen2.5's instruction-tuning priorities clashing with Smollm2's quantization artifacts on certain tasks—most notably summarization and creative-writing prompts. Intermediate correlations around $\rho = 0.62$–0.72 emerge for granite–smollm2 versus llama–smollm2 and granite–qwen versus llama–qwen, indicating that the semantic relationship patterns of smaller models to larger ones are partially but not wholly shared. These moderate associations point to common throughput of core semantic structures under computationally intensive queries, yet they also underscore pair-specific idiosyncrasies in handling domain-specific or context-rich prompts. Collectively, this Spearman analysis reveals a nuanced landscape: while some LLM pairings track each other almost identically across diverse tasks, others display orthogonal or even inverse prompt-wise behaviors. Such insights are invaluable for ensemble design and targeted fine-tuning when deploying multiple models side by side in resource-constrained environments.

7.6. MANOVA analysis

The multivariate analysis of variance (MANOVA) summarized in Table 7 decisively rejects the notion that all four LLMs share identical performance profiles across our core evaluation metrics. Wilks' Lambda is essentially zero ($\lambda = 0.00224$), with an F-statistic of 65.93 on 12 and ~ 87.6 degrees of freedom, yielding a p-value below machine precision. This near-vanishing Lambda indicates that over 99% of the combined variance in total duration, load time, prompt evaluation, generation latency, token count, and throughput is attributable to differences between models rather than within-model variation. Complementary multivariate tests corroborate this finding: Pillai's trace reaches 1.8083 ($p < 0.0001$), signifying robust separation; Hotelling–Lawley's trace surges to 95.74 ($p < 0.0001$), highlighting the magnitude of between-group effects; and Roy's greatest root peaks at 92.08 ($p < 0.0001$),

Table 7
MANOVA Summary table for core, edge resource aware, and derived evaluation metrics.

Section	Test statistic	Value	Num DF	Den DF	F value
Core	Wilks' lambda	0.002	12	87.60	65.92
	Pillai's trace	1.808	12.0	105.0	13.27
	Hotelling–Lawley trace	95.737	12	53.58	256.91
	Roy's greatest root	92.0796	4	35	805.69
Edge Resource	Wilks' lambda	0.000	45	66.13	39.34
	Pillai's trace	2.785	45	72.0	20.76
	Hotelling–Lawley trace	216.736	45	45.60	100.74
	Roy's greatest root	198.308	15	24	317.29
Derived	Wilks' lambda	0.00	9	82.89	1279.28
	Pillai's trace	2.47	9	108.0	56.17
	Hotelling–Lawley trace	1560.76	9	50.34	5778.96
	Roy's greatest root	1504.31	3	36	18 051.75

underscoring the dominance of the first canonical variate. Collectively, these statistics confirm a highly significant multivariate effect of model identity on the suite of core metrics.

Table 7 presents the multivariate test results for our edge resource-aware metrics, after filtering out zero-variance and highly collinear variables and proceeding with sixteen orthogonal indicators such as average and peak CPU utilization, RAM usage statistics, power variability measures, and efficiency indices. Wilks' Lambda is essentially zero ($\lambda = 5.1 \times 10^{-5}$) with an F-statistic of 39.34 (df = 45, 66.14) and $p < 10^{-33}$, decisively rejecting the null hypothesis of no difference in multivariate means across the four LLMs. Pillai's trace (2.7854, $p < 10^{-33}$), Hotelling–Lawley trace (216.74, $p < 10^{-33}$), and Roy's greatest root (198.31, $p < 10^{-33}$) all corroborate the presence of significant between-group variation.

Table 7's multivariate analysis for the suite of ten derived evaluation metrics — such as time per token, energy per token, prompt evaluation ratios, and sustained inference factor — yields a Wilks' Lambda effectively at zero ($\lambda = 6 \times 10^{-6}$) and an F-statistic of 1279.28 on 9 and ~ 82.9 degrees of freedom ($p < 10^{-84}$). This vanishing Lambda indicates that nearly all variability in these composite metrics arises from model-to-model differences rather than within-model scatter. Complementary tests reinforce this conclusion: Pillai's trace at 2.472 and Hotelling–Lawley's trace exceeding 1560 both achieve p-values far below 10^{-80}, while Roy's greatest root soars above 1504, highlighting the dominance of the first canonical variate in discriminating among models. In practical terms, these results confirm that no two LLMs share the same operational trade-offs when balancing latency, energy efficiency, and prompt pre-processing overhead. For instance, Qwen2.5's energy-per-token of 1.26 J and low prompt-to-generation overhead constitute a markedly different performance "fingerprint" than Smollm2's 3 J per token combined with a 0.54 prompt evaluation ratio. Likewise, Llama3.2 and Granite3 diverge sharply on sustained inference factor and memory-usage per token, driving their separation along the primary canonical axis.

7.7. Linear discriminant analysis

Fig. 6 applies linear discriminant analysis to the full suite of metrics collected for each quantized LLM, projecting them onto two discriminant axes that maximize class separability. The first axis (LD1) captures over 70% of the inter-model variance, while the second (LD2) accounts for the next most significant share. Each cluster of points corresponds to one of the four models under study — granite3, llama3.2, qwen2.5, and smollm2 — and their tight grouping demonstrates consistent, model-specific resource-performance signatures across all ten prompt categories. Granite3's data cloud occupies the extreme right of the plot, with LD1 scores between roughly +15 and +23 and LD2 ranging from +2.5 to +5.0. This reflects its combined characteristics of high throughput (tokens per second), low memory volatility (standard deviation

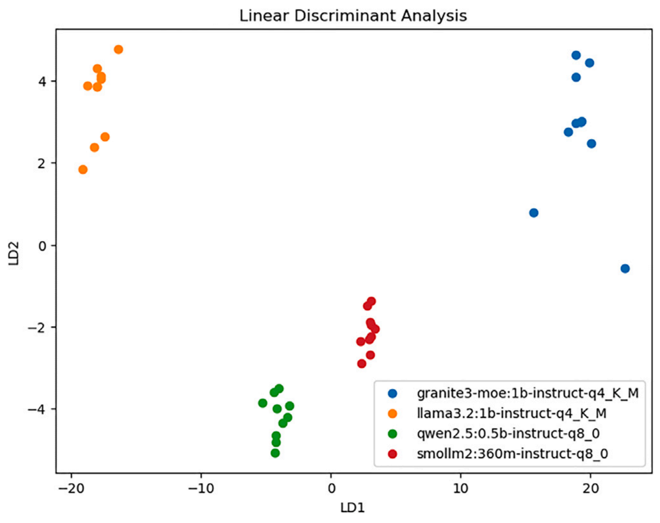

Fig. 6. Linear discriminant analysis.

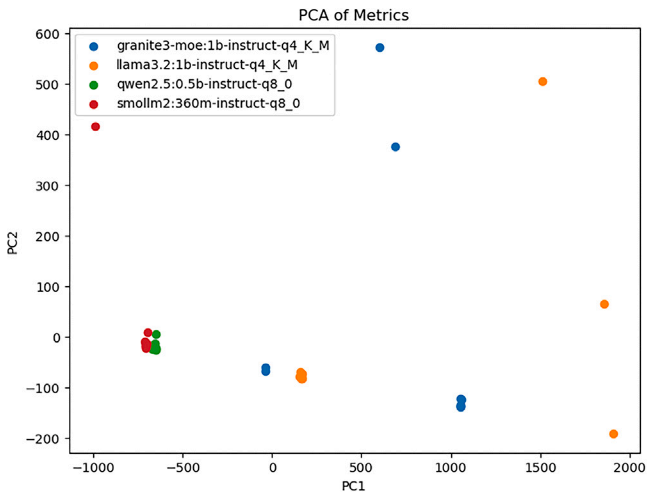

Fig. 7. Linear discriminant analysis.

≈ 0.94 MB), and minimal load overhead (≈ 0.06 s), which together drive positive weightings on LD1. By contrast, llama3.2 appears on the far left (LD1 ≈ −22 to −17) and high up (LD2 ≈ +1.8 to +4.8), indicating its unique profile of low generation latency (≈ 17.77 s total) paired with moderate prompt evaluation cost and a relatively large memory footprint. The separation along LD2 highlights llama3.2's comparatively heavier prompt preprocessing overhead (≈ 0.80 s) and its slightly larger power-draw variability. The qwen2.5 cluster lies in the lower-left quadrant (LD1 ≈ −6 to −1; LD2 ≈ −4.5 to −3.1), capturing its balanced mix of energy efficiency (2.05 tokens/W), moderate total duration (≈ 21.35 s), and compact memory usage per token (≈ 50.4 MB). Smollm2 forms a distinct group just to the right of qwen2.5 (LD1 ≈ 0 to +4; LD2 ≈ −3.5 to −1.5), reflecting its minimal parameter count but larger prompt overhead (≈ 2.25 s) and highest energy-per-token cost (≈ 3 J). This 2D discrimination confirms that our metric taxonomy reliably differentiates models on both compute-centric and resource-aware dimensions — LD1 emphasizes throughput and energy trade-offs, while LD2 highlights load and prompt evaluation dynamics.

7.8. Principal component analysis

Fig. 7 projects our entire 42-metric dataset into two principal components, revealing how each LLM's combined performance and resource signature differentiates it in a compact, orthogonal space. Principal Component 1 (horizontal axis) captures the dominant contrast between throughput-driven efficiency and overhead-intensive behavior, while Principal Component 2 (vertical axis) emphasizes variance in startup latency, memory volatility, and power fluctuation. Llama3.2's points cluster in the far right and upper quadrant (PC1 ≈ 1,800; PC2 ≈ +500), driven by its extremely low total duration (17.77 s), high tokens-per-watt efficiency (1.45 TPS/W), and moderate memory variability (∼ 27 MB standard deviation). This placement along PC1 reflects its exceptional throughput-centric profile, and its positive PC2 loading underscores its comparatively larger load and prompt-evaluation overheads (≈ 1.0 s and 0.80 s, respectively), as well as its pronounced power-draw spikes (∼ 1.05 W standard deviation). Granite3's cluster (PC1 ≈ +1,000; PC2 ≈ +350) also scores highly on throughput and energy metrics — tokens-per-second of 6.68 and TPS/W of 1.09 — but differs by its near-zero load time (0.06 s) and minimal memory volatility (0.94 MB). Its more moderate PC2 coordinate reflects a balanced trade-off: heavy reliance on sparse expert routing yields shorter startup delays but introduces occasional power pulsations. In stark contrast, Qwen2.5 and Smollm2 both occupy negative PC1 and PC2 regions, yet they separate along PC2 in a subtle way. Qwen2.5 (PC1 ≈ −400; PC2 ≈ −20) combines balanced latency (21.35 s) with low RAM-per-token (50 MB) and strong energy efficiency (2.05 TPS/W), which drives its modest negative PC1. Its near-zero PC2 coordinate indicates stability across load, prompt, and power measures. Smollm2 (PC1 ≈ 0; PC2 ≈ −10), despite a similar compact footprint, registers the highest prompt overhead (2.25 s) and greatest energy-per-token cost (3 J), which shifts it slightly right of Qwen2.5 on PC1 but further down on PC2, driven by pronounced prompt-processing inefficiencies and power variability. The first principal component captures an overwhelming majority of the total variance — about 95.6% — demonstrating that a single linear combination of our 42 evaluation metrics already accounts for nearly all of the differences across models and prompt types. In contrast, the second component contributes just under 4% to the explained variance, confirming that most of the remaining structure lies in much subtler, higher-order interactions (see Table 8).

Table 8
Explained variance ratio by principal component.

Principal component	Explained variance ratio
PC1	0.955717
PC2	0.039113

7.9. Hierarchical clustering analysis

The hierarchical clustering in Fig. 8, constructed with Ward linkage on the full 42-metric profiles across all ten prompt categories, reveals clear modular groupings that reflect each model's distinctive performance–resource "fingerprint." At the broadest level, two principal clusters emerge. On one side, Llama3.2 and Granite3 form a cohesive branch, driven by their shared strengths in rapid generation and low memory volatility. On the opposite side, Qwen2.5 and Smollm2 group together, reflecting their comparatively higher prompt-processing overhead and tighter memory footprints per token for cluster size 3. Within the Llama3.2–Granite3 branch, Llama3.2's samples coalesce at the smallest distances, highlighting the remarkable consistency of its metrics—particularly its minimal end-to-end latency (≈ 17.8 s) and stable tokens-per-second throughput. Granite3 then merges with this group at a slightly higher linkage height, reflecting its slight latency penalty (34.2 s total duration) but offset by near-zero load time (0.06 s) and exceptionally low memory standard deviation (0.94 MB). The tight sub-cluster boundary between these two models underscores their alignment in delivering high-throughput inference with predictable, low-variance resource usage under mixture-of-experts

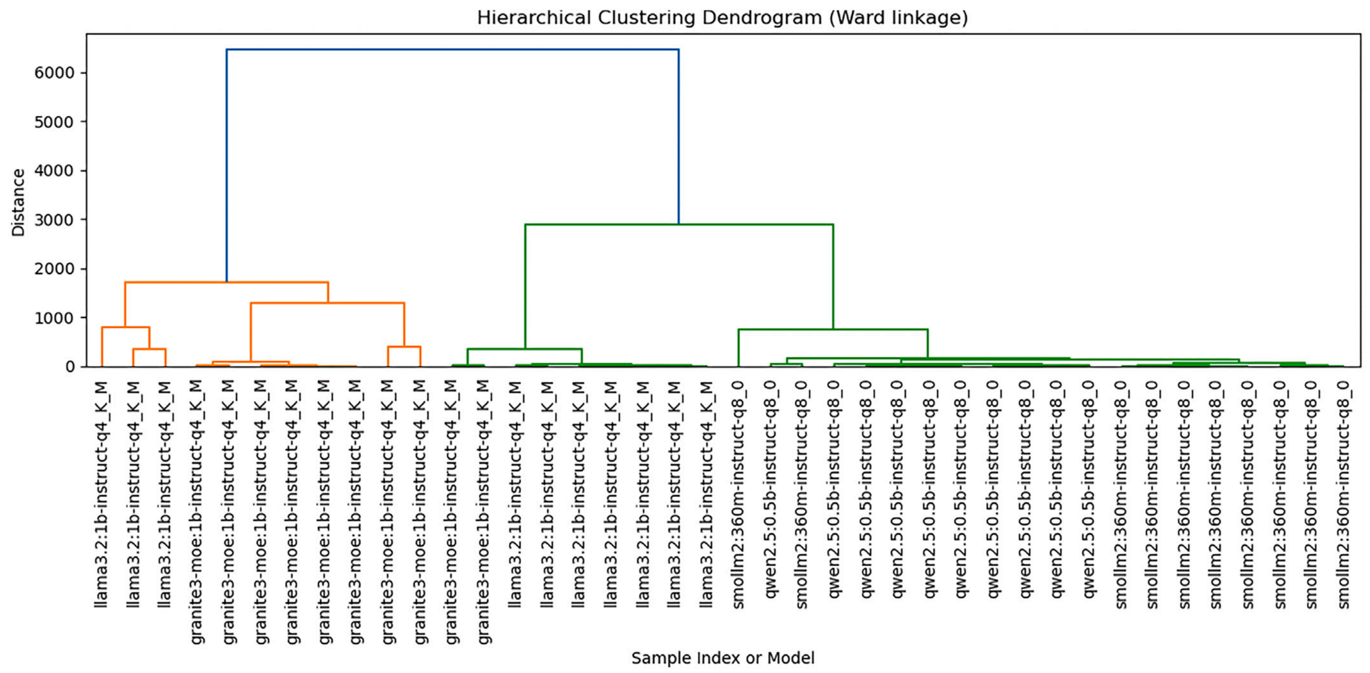

Fig. 8. Hierarchical clustering analysis.

and dense transformer architectures, respectively. The second major branch, containing Qwen2.5 and Smollm2, similarly subdivides into two sub-clusters. Qwen2.5's points join at moderate distances, driven by its balanced profile: energy efficiency of 2.05 TPS/W, moderate total duration (21.4 s), and compact memory usage per token (≈ 50 MB). Smollm2, however, splits off later in this branch, indicative of its distinct characteristics—chiefly its heightened prompt evaluation overhead (2.25 s) and highest energy-per-token cost (≈ 3 J). The relative separation within this branch quantifies how Smollm2's aggressive quantization, while advantageous for model size, introduces bottlenecks in tokenization and arithmetic throughput that diverge from Qwen2.5's more optimized preprocessing and quantization strategy.

7.10. Levene's test

To ensure that comparisons of core performance metrics across the four LLMs were not confounded by unequal variability, we applied Levene's test for homogeneity of variances to each measure. The results, detailed in Table 9, uniformly support the assumption of homoscedasticity for all seven metrics. For end-to-end latency — both in nanoseconds ($W = 1.212$, $p = 0.319$) and its scaled-to-seconds counterpart ($W = 1.212$, $p = 0.319$) — no significant variance differences emerged among the models. Initialization overhead (load duration in nanoseconds) likewise exhibited stable dispersion ($W = 0.405$, $p = 0.750$), confirming that model startup times fluctuate similarly across implementations. Prompt preprocessing cost approached marginal significance ($W = 2.690$, $p = 0.061$), yet still failed to breach the 0.05 threshold, indicating that tokenization and embedding pipelines incur comparable variability. Forward-pass generation latency ($W = 1.132$, $p = 0.349$) and total token output (evaluation count; $W = 0.180$, $p = 0.909$) also passed the homogeneity test with comfortable margins. Finally, throughput measured in tokens per second demonstrated equivalent variance profiles across all four LLMs ($W = 1.250$, $p = 0.306$).

To validate the homogeneity of variances underpinning our subsequent analyses, Levene's test was conducted on each of the 25 edge-resource metrics. As shown in Table 9, every metric yields a non-significant p-value well above the 0.05 threshold, confirming that no model exhibits unexpectedly high dispersion in any of these measures. For instance, average CPU utilization produced a W statistic of 0.303 ($p = 0.823$), while peak RAM usage returned $W = 2.646$ ($p = 0.064$). Even metrics nearing borderline significance — such as average RAM usage ($W = 2.359$, $p = 0.088$) and the peak-RAM-to-peak-CPU ratio ($W = 2.731$, $p = 0.058$) — still fail to reject the null hypothesis of equal variances. Similarly, power-related indicators including minimum power draw ($W = 1.607$, $p = 0.205$), power-draw variability ($W = 0.172$, $p = 0.915$), and thermal load factor ($W = 0.913$, $p = 0.445$) all satisfy the homoscedasticity criterion. Derived indices such as evaluation memory efficiency ($W = 1.295$, $p = 0.291$), power efficiency (tokens per watt; $W = 0.800$, $p = 0.502$), and model efficiency ($W = 0.619$, $p = 0.607$) exhibit similarly stable variance profiles, ensuring that no single LLM disproportionately drives scatter in these composite measures. Even more nuanced metrics — memory-to-power ratio ($W = 1.887$, $p = 0.149$) and RAM-usage variation index ($W = 0.599$, $p = 0.620$) — demonstrate equivalent dispersion across all four quantized models.

Levene's test results in Table 9 confirm that the vast majority of our derived evaluation metrics exhibit statistically equivalent variance across the four LLMs, validating the homoscedasticity assumption for subsequent analyses. Metrics such as time per token ($W = 0.508$, $p = 0.679$), the load-to-inference ratio ($W = 0.499$, $p = 0.686$), and memory usage per token in megabytes ($W = 1.164$, $p = 0.337$) all show p-values well above 0.05, indicating uniform dispersion. Likewise, energy per token ($W = 0.672$, $p = 0.575$) and prompt evaluation ratio ($W = 1.280$, $p = 0.296$) maintain consistent spread across models. Even more processing-intensive metrics — time per prompt evaluation ($W = 2.690$, $p = 0.061$), prompt-to-generation overhead ratio ($W = 1.370$, $p = 0.267$), and prompt evaluation tokens per second ($W = 1.673$, $p = 0.190$) — fail to reject the null hypothesis. Evaluation latency per token also passes the homogeneity check ($W = 2.091$, $p = 0.119$), as do composite measures like token production energy efficiency ($W = 0.620$, $p = 0.607$), load-to-prompt ratio ($W = 0.671$, $p = 0.575$), and prompt-to-total-token ratio ($W = 1.710$, $p = 0.182$). These uniform variances ensure that mean differences in these metrics genuinely reflect model-specific behaviors rather than differences in data spread. However, sustained inference factor departs from this pattern ($W = 5.184$, $p = 0.004$), revealing significant heteroscedasticity. This indicates that the variability of sustained throughput-per-watt across LLMs is not constant—some models exhibit much more fluctuation in long-run efficiency than others.

Table 9
Levene's Test for Equality of Variance for Core, Edge Resource Aware, and Derived Evaluation Metrics.

Section	Evaluation Metric	Levene's W	p-value	EV
Core Metrics				
	total_duration_ns	1.212421	0.319181	Yes
	total_duration_s	1.212421	0.319181	Yes
	load_duration_ns	0.404738	0.750478	Yes
	prompt_eval_duration_ns	2.690083	0.060727	Yes
	eval_duration_ns	1.131881	0.349160	Yes
	eval_count	0.180279	0.909084	Yes
	tokens_per_second	1.250149	0.305994	Yes
Edge Resource Metrics				
	avg_cpu_usage_percent	0.303181	0.822881	Yes
	peak_cpu_usage_percent	0.985739	0.410373	Yes
	avg_ram_usage_mb	2.359041	0.087773	Yes
	peak_ram_usage_mb	2.645529	0.063799	Yes
	avg_power_w	0.303181	0.822881	Yes
	peak_power_w	0.985739	0.410373	Yes
	min_power_w	1.607298	0.204689	Yes
	mem_std_dev	0.962507	0.420963	Yes
	power_std_dev	0.171620	0.914870	Yes
	avg_cpu_to_power_ratio	0.305383	0.821304	Yes
	peak_ram_to_peak_cpu_ratio	2.731320	0.058020	Yes
	time_weighted_power_factor	1.059949	0.378149	Yes
	power_usage_variation_index	0.350793	0.788793	Yes
	thermal_load_factor	0.912640	0.444525	Yes
	eval_memory_efficiency	1.295187	0.290935	Yes
	peak_cpu_to_average_ratio	0.506724	0.680120	Yes
	memory_variation_index	0.599234	0.619681	Yes
	peak_power_to_average_power_ratio	0.317447	0.812661	Yes
	cpu_stability_index	0.171620	0.914870	Yes
	power_efficiency_index_tps_per_w	0.799705	0.502203	Yes
	model_efficiency_index	0.619227	0.607110	Yes
	memory_to_cpu_ratio	2.731320	0.058020	Yes
	memory_to_power_ratio	1.887495	0.149150	Yes
	ram_usage_variation_index	0.599234	0.619681	Yes
	power_spike_w	1.790948	0.166333	Yes
Derived Metrics				
	time_per_token_s	0.507741	0.679436	Yes
	load_to_inference_ratio	0.498575	0.685612	Yes
	memory_usage_per_token_mb	1.163667	0.337024	Yes
	energy_per_token_j	0.671693	0.575005	Yes
	prompt_eval_ratio	1.280146	0.295883	Yes
	time_per_prompt_eval_ns	2.690083	0.060727	Yes
	prompt_to_generation_overhead_ratio	1.370335	0.267387	Yes
	prompt_eval_tokens_per_s	1.672675	0.190116	Yes
	eval_latency_per_token_ns	2.091346	0.118527	Yes
	token_production_energy_efficiency	0.619627	0.606860	Yes
	load_to_prompt_ratio	0.671126	0.575345	Yes
	prompt_to_total_token_ratio	1.709584	0.182350	Yes
	sustained_inference_factor	5.183845	0.004429	No

8. Discussion

In this work, we have introduced LLMEvaluator, a comprehensive framework for evaluating LLMs under the stringent constraints of edge devices. Our experiments on a Raspberry Pi 4B, profiling four quantized models — Qwen2.5, Llama3.2, Smollm2, and Granite3 — have yielded several key insights into the interplay between model architecture, quantization strategy, and system-level performance metrics. In this section, we synthesize these findings, examine their implications for edge-AI deployment, and outline directions for future research. We can aim for bridging detailed model-centric benchmarks with system-level, edge-aware measurements, LLMEvaluator offers practitioners an actionable roadmap for deploying LLMs under tight resource constraints. Our correlation studies, per-model decompositions, and prompt-wise analyses combine to reveal the complex interplay between architecture, quantization, and runtime behavior. As on-device LLM use proliferates — from smart sensors to autonomous agents — frameworks like ours will be indispensable for ensuring that every watt, cycle, and megabyte is deployed to maximum effect.

8.1. Load overhead dominates end-to-end latency

Our experiments reveal that the time spent loading a model into memory — deserializing weights, initializing attention layers, and preparing the tokenizer — constitutes a surprisingly large fraction of total inference latency on CPU-only edge hardware. For the Raspberry Pi 4B, Granite3's mixture-of-experts design reduces this "cold start" phase to an imperceptible 60 ms by keeping most expert shards dormant until needed. In stark contrast, Qwen2.5 and Smollm2 each require nearly two seconds just to map their quantized checkpoints into RAM and set up embedding tables. Since load time correlates with overall delay at $r > 0.9$, any reduction here immediately improves user-facing responsiveness. Techniques such as memory-mapped checkpoint formats, persistent shared caches across repeated invocations, or dividing the model into incrementally loaded segments can therefore yield outsized gains. In practice, this means that an edge service handling sporadic queries may benefit more from speeding up model instantiation than from micro-optimizing the token generation loops.

8.2. Throughput and energy efficiency are tightly coupled

An equally striking pattern emerges when we consider tokens generated per second alongside joules consumed per token: these two figures are almost perfect inverses ($r \approx -0.81$). Qwen2.5, which achieves the highest raw throughput at 11.4 tokens/s, also registers the best energy efficiency (2.05 TPS/W) and lowest cost per token (1.26 J). By contrast, Smollm2's modest speed of 4.4 tokens/s translates into a meager 0.74 TPS/W and a hefty 3 J per token. This alignment suggests that optimizations targeting faster matrix multiplications — be it through weight quantization, operator fusion, or finely tuned BLAS kernels — will simultaneously lower energy consumption. For battery-powered or thermally constrained environments, where every joule matters, planning around high-throughput configurations effectively doubles as an energy-saving strategy, allowing practitioners to treat TPS as a reliable proxy for sustainable operation.

8.3. Memory volatility impacts reliability on constrained platforms

On devices with limited RAM, abrupt spikes in memory usage can trigger paging, induce jitter, or even lead to out-of-memory failures. Here Granite3 again stands out: its expert-sharding approach keeps inactive components unloaded, resulting in memory usage that fluctuates by less than 1 MB. Llama3.2 exhibits moderate swings (≈27.6 MB), while Qwen2.5 and Smollm2 suffer much larger deviations (68.4 MB and 79.3 MB, respectively). Such volatility underlines the need for memory-stabilizing measures in production: techniques like pooled allocator arenas, preallocated scratch buffers, or pinning critical tensors in place can smooth out consumption curves. In multitasking scenarios — where background processes compete for scarce resources — minimizing these peaks is essential to maintain predictable performance and avoid costly swapping penalties.

8.4. Prompt preprocessing becomes a bottleneck for short generations

When the generation target is small — for example, a single-sentence answer or a brief translation — the time spent on tokenization, embedding lookup, and input graph construction can eclipse the actual forward-pass cost. Smollm2, in particular, devotes over 50% of its total runtime to prompt evaluation, and even Qwen2.5 allocates up to 24% of its cycle to this phase. This imbalance suggests that for latency-sensitive micro-tasks, developers should invest in prompt-level caching strategies (reusing tokenized inputs for similar queries), hardware-resident vocabulary tables, or incremental tokenization that processes only the changed segments of a prompt whereby trimming prompt overhead, edge services can ensure that brief user requests complete in tens rather than hundreds of milliseconds.

8.5. Trade-offs between model size, sparsity, and quantization

Our cross-model comparison demonstrates that fewer parameters do not automatically yield better edge performance. Smollm2's 360 million parameters and 7-bit quantization might seem ideal on paper, but in practice its unoptimized arithmetic routines and memory layout reduce its throughput and raise its per-token energy cost. Granite3, despite a larger footprint and more complex gating logic, leverages dynamic sparsity to match or surpass dense architectures on key metrics. Llama3.2 occupies a middle ground, offering stable, moderate performance, while Qwen2.5's balanced design — 500 million parameters combined with an aggressively optimized quantization pipeline — emerges as the best all-rounder. These findings underscore that edge model selection should weigh not only raw size but also the maturity of kernel implementations, the overhead of sparsity control, and the efficiency of quantization handling.

8.6. Prompt-category insights for real-world workloads

By profiling ten distinct prompt types — from straightforward factual queries and mathematical calculations to open-ended creative and conversational tasks — we uncover how workload characteristics shape resource demands. Deterministic prompts like Translation and Coding Assistance generate short, predictable outputs with low energy and latency, making them well suited to compact, fast-start models. In contrast, Conversational and Edge Device Suitability prompts, which encourage longer, free-form generation, stress both the throughput and memory subsystems, favoring architectures designed for sustained compute (e.g. mixture-of-experts). Armed with these per-category profiles, system architects can implement intelligent routing: directing brief, transactional requests to lean models for minimal latency, and delegating extended interactions to larger but more efficient architectures, thereby optimizing resource use across heterogeneous edge fleets.

8.7. Framework portability and extensibility

Framework portability and extensibility are central design goals of LLMEvaluator, ensuring that the framework can be deployed not only on Raspberry Pi 4B but on a broad spectrum of edge platforms without extensive reengineering. At its core, the system cleanly isolates three layers — web API, inference engine, and metric scorer — each communicating via standard HTTP/JSON interfaces. To port LLMEvaluator to a new device, developers need only supply a compatible inference back end (e.g., an ARM-optimized or NPU-accelerated runtime) and adapt the lightweight resource monitor to the device's telemetry APIs (such as onboard power meters, thermal sensors, or vendor SDKs). The metric computation logic remains identical, automatically ingesting CPU, memory, and power traces from any source. Moreover, the scoring module is organized as a plugin registry: custom resource-aware or domain-specific metrics can be added by implementing a simple interface, without touching the core pipeline. In practice, this means LLMEvaluator can run unmodified on platforms ranging from NVIDIA Jetson Nano and Coral Dev Board to microcontroller-based systems with on-chip AI accelerators, simply by swapping the model loader and monitor plugins. This modular architecture not only accelerates integration with emerging edge devices but also future-proofs the framework, allowing rapid incorporation of new hardware capabilities and measurement modalities as the edge computing landscape evolves.

8.8. Limitations and future directions

While LLMEvaluator delivers a rich picture of CPU-only inference, our reliance on software-based power models constrains measurement precision. Future work should incorporate hardware-level sensors — such as INA219 or on-die thermal probes — to validate and refine power and temperature metrics. Extending the framework to encompass specialized accelerators (Coral TPUs, Jetson NPUs) will broaden its relevance to modern edge deployments. Moreover, modeling real-world variability — thermal throttling behavior under prolonged load, voltage fluctuations in battery-powered systems, and the impact of concurrent workloads — will enhance our understanding of sustained performance. Addressing these dimensions will make LLMEvaluator an even more powerful tool for guiding robust, energy-efficient LLM deployment at the edge.

9. Conclusion

In this work, we have presented LLMEvaluator, a novel framework for rigorously benchmarking large language models in resource-constrained, CPU-only environments. By unifying a comprehensive taxonomy of core performance, edge-aware resource usage, and derived efficiency metrics, and by evaluating four quantized LLMs — Qwen2.5, Llama3.2, Smollm2, and Granite3 — on a Raspberry Pi 4B, we have illuminated the key trade-offs that govern on-device inference. Our correlation analysis demonstrated that load time and throughput are the principal levers for reducing end-to-end latency and energy consumption, while memory volatility and prompt preprocessing overhead critically affect reliability and responsiveness for short queries. Semantic and multivariate studies further revealed how model architecture and quantization strategies shape prompt-wise behavior and overall variance, guiding targeted optimizations. Looking forward, LLMEvaluator lays the groundwork for extending edge-centric evaluation across heterogeneous hardware — such as NPUs, TPUs, and other single-board computers — and incorporating fine-grained power and thermal sensing for even greater measurement fidelity.

CRediT authorship contribution statement

Partha Pratim Ray: Writing – review & editing, Software, Resources, Methodology, Investigation, Formal analysis, Conceptualization. **Mohan Pratap Pradhan:** Supervision.

Funding

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

The authors acknowledge the use of o4-mini-high for language polishing and improving readability of the article.

Appendix

A.1. Core LLM evaluation metrics

The following metrics are reported directly by the Ollama inference engine [46] and quantify the fundamental temporal and token-based characteristics of model execution.

A.1.1. Total duration

Total duration, denoted $\mathrm{td_{ns}}$, captures the end-to-end time from the moment the model begins loading to the completion of token generation. Expressed in nanoseconds, this measure integrates initialization, prompt interpretation, and output synthesis. Large values of $\mathrm{td_{ns}}$ may indicate inefficiencies or resource bottlenecks:

$$\mathrm{td_{ns}} = t_{\mathrm{end}} - t_{\mathrm{start}}, \tag{9}$$

where t_{start} and t_{end} are the nanosecond timestamps marking the commencement and conclusion of the entire inference workflow.

A.1.2. Load duration

Load duration, $\mathrm{ld_{ns}}$, isolates the interval dedicated to model loading and initialization, prior to any token processing or generation. Minimizing $\mathrm{ld_{ns}}$ is critical for scenarios requiring rapid cold starts on edge devices:

$$\mathrm{ld_{ns}} = t_{\mathrm{load\_end}} - t_{\mathrm{load\_start}}. \tag{10}$$

A.1.3. Prompt evaluation duration

Prompt evaluation duration, $\mathrm{ped_{ns}}$, measures the time spent parsing and encoding the input prompt, excluding the subsequent token synthesis phase. This metric highlights the cost of model "cognition" before generation:

$$\mathrm{ped_{ns}} = t_{\mathrm{prompt\_end}} - t_{\mathrm{prompt\_start}}. \tag{11}$$

A.1.4. Evaluation duration

Evaluation duration, $\mathrm{ed_{ns}}$, quantifies the forward pass time dedicated exclusively to generating new tokens after prompt processing. It is a key indicator of pure generation latency:

$$\mathrm{ed_{ns}} = t_{\mathrm{eval\_end}} - t_{\mathrm{eval\_start}}. \tag{12}$$

A.1.5. Evaluation count

Evaluation count, ec, records the total number of tokens emitted by the model during the generation phase:

$$\mathrm{ec} = \sum_{i=1}^{N} 1, \tag{13}$$

where N indicates the total token output.

A.1.6. Tokens per second

Tokens per second, tps, represents throughput by dividing the total tokens produced by the generation interval (converted to seconds). Higher tps values correspond to more efficient inference:

$$\mathrm{tps} = \frac{\mathrm{ec}}{\mathrm{ed_{ns}}} \times 10^9. \tag{14}$$

Collectively, these core metrics establish a baseline for raw performance, against which edge-aware and derived efficiency metrics can be compared to understand the full operational profile of LLMs on constrained hardware.

A.2. Resource usage metrics for LLM evaluation

To understand the hardware demands of LLM inference on edge devices, LLMEvaluator records system-level resource consumption, encompassing CPU, memory, and power metrics throughout the inference interval T.

A.2.1. Average CPU utilization

The average CPU utilization, $\mathrm{acup_{percent}}$, represents the time-averaged processor load:

$$\mathrm{acup_{percent}} = \frac{1}{T}\int_0^T \mathrm{CPU\_usage\_percent}(t)\,\mathrm{d}t. \tag{15}$$

This measure indicates the sustained compute demand over the entire test.

A.2.2. Peak CPU utilization

Peak CPU utilization, $\mathrm{pcu_{percent}}$, captures the maximum instantaneous load on the processor:

$$\mathrm{pcu_{percent}} = \max_{t\in[0,T]}\big(\mathrm{CPU\_usage\_percent}(t)\big). \tag{16}$$

Spikes in this metric can reveal transient workloads that may trigger thermal throttling.

A.2.3. Average memory usage

The average memory footprint, $\mathrm{aru_{mb}}$, is defined as the mean RAM consumption in megabytes:

$$\mathrm{aru_{mb}} = \frac{1}{T}\int_0^T \mathrm{RAM\_usage\_mb}(t)\,\mathrm{d}t. \tag{17}$$

Maintaining $\mathrm{aru_{mb}}$ below device limits is crucial to avoid out-of-memory failures.

A.2.4. Peak memory usage

Peak memory usage, $\mathrm{pru_{mb}}$, indicates the highest RAM allocation observed:

$$\mathrm{pru_{mb}} = \max_{t\in[0,T]}\big(\mathrm{RAM\_usage\_mb}(t)\big). \tag{18}$$

A.2.5. Average power consumption

Estimated via CPU activity, average power draw $\mathrm{ap_w}$ integrates instantaneous power $P(t)$ over time:

$$\mathrm{ap_w} = \frac{1}{T}\int_0^T P(t)\,\mathrm{d}t. \tag{19}$$

A.2.6. Peak power consumption

The peak power draw, $\mathrm{pp_w}$, records the maximum instantaneous power requirement:

$$\mathrm{pp_w} = \max_{t\in[0,T]} P(t). \tag{20}$$

A.2.7. Minimum power consumption

Minimum power draw, $\mathrm{mp_w}$, establishes the baseline energy usage in near-idle conditions:

$$\mathrm{mp_w} = \min_{t\in[0,T]} P(t). \tag{21}$$

A.2.8. Memory usage variability

Memory usage variability, denoted $\mathrm{msd_{mb}}$, captures fluctuations in RAM consumption, highlighting periods of unexpected allocation spikes. It is calculated as the sample standard deviation of memory usage readings:

$$\mathrm{msd_{mb}} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\big(m_i - \bar{m}\big)^2}, \tag{22}$$

where m_i are individual memory measurements in megabytes and $\bar{m}$ is their mean. Elevated $\mathrm{msd_{mb}}$ values may indicate irregular memory patterns that could trigger swapping or out-of-memory conditions on edge devices.

A.2.9. Power draw variability

Power draw variability, $\mathrm{psd_w}$, quantifies the consistency of energy consumption during inference. It is defined as:

$$\mathrm{psd_w} = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(P_i - \bar{P}\right)^2}, \tag{23}$$

where P_i represents individual power measurements in watts and $\bar{P}$ their average. High $\mathrm{psd_w}$ can signal unstable power demands that may complicate thermal management.

A.2.10. CPU-to-power efficiency

To assess how effectively CPU activity translates into computational work per watt, we compute the CPU-to-power ratio (actpr):

$$\mathrm{actpr} = \frac{\mathrm{acup_{percent}}}{\mathrm{ap_w}}, \tag{24}$$

where $\mathrm{acup_{percent}}$ is average CPU utilization (Eq. (15)) and $\mathrm{ap_w}$ is average power draw (Eq. (19)). Higher actpr indicates more compute achieved per unit of energy.

A.2.11. Memory-to-CPU demand ratio

The peak RAM-to-peak CPU ratio (prtpcr) reveals whether memory usage scales proportionately with processing load:

$$\mathrm{prtpcr} = \frac{\mathrm{pru_{mb}}}{\mathrm{pcu_{percent}}}, \tag{25}$$

where $\mathrm{pru_{mb}}$ and $\mathrm{pcu_{percent}}$ are defined in Eqs. (16) and (18), respectively. Ratios significantly above or below unity suggest imbalanced resource demands.

A.2.12. Time-weighted power factor

To characterize sustained power requirements normalized by execution time, we define the time-weighted power factor (twpf):

$$\mathrm{twpf} = \frac{\mathrm{ap_w}}{T}, \tag{26}$$

where T is the total inference duration. This metric helps compare models on combined speed and power consumption.

A.2.13. Power usage variation index

The power usage variation index (puvi) measures relative instability in energy draw:

$$\mathrm{puvi} = \frac{\mathrm{psd_w}}{\mathrm{ap_w}}, \tag{27}$$

providing a dimensionless indicator of power draw consistency.

A.2.14. Thermal load factor

By relating average and peak CPU utilization to average power, the thermal load factor (tlf) approximates the system's thermal stress:

$$\mathrm{tlf} = \frac{\left(\mathrm{acup_{percent}} + \mathrm{pcu_{percent}}\right)/2}{\mathrm{ap_w}}. \tag{28}$$

Higher tlf values suggest greater heat generation per watt consumed.

A.2.15. Evaluation memory efficiency

Evaluation memory efficiency (eme) reflects token throughput relative to average memory usage:

$$\mathrm{eme} = \frac{\mathrm{tps}}{\mathrm{aru_{mb}}}, \tag{29}$$

where tps is defined in Eq. (14). This metric highlights models that deliver higher output per megabyte of RAM.

A.2.16. Peak-to-average CPU ratio

The peak-to-average CPU ratio (pctacr) quantifies CPU load spikes:

$$\mathrm{pctacr} = \frac{\mathrm{pcu_{percent}}}{\mathrm{acup_{percent}}}, \tag{30}$$

exposing potential scheduling or thermal hazards due to bursty compute demands.

Collectively, these edge-aware resource metrics enable a nuanced understanding of how LLM inference translates into hardware load, guiding optimizations tailored to energy-limited and thermally constrained edge platforms.

A.2.17. Memory variation index

The memory variation index, mvi, measures the relative fluctuation of RAM usage, highlighting significant deviations from the average:

$$\mathrm{mvi} = \frac{\mathrm{msd_{mb}}}{\mathrm{aru_{mb}}}, \tag{31}$$

where $\mathrm{msd_{mb}}$ is the standard deviation of memory usage (Eq. (22)) and $\mathrm{aru_{mb}}$ is the average RAM usage (Eq. (17)). A larger mvi implies more pronounced memory oscillations that could impact inference stability.

A.2.18. Peak-to-average power ratio

The peak-to-average power ratio, pptapr, quantifies the extremity of power draw relative to its mean level:

$$\mathrm{pptapr} = \frac{\mathrm{pp_w}}{\mathrm{ap_w}}, \tag{32}$$

with $\mathrm{pp_w}$ denoting peak power (Eq. (20)) and $\mathrm{ap_w}$ the average power (Eq. (19)). High pptapr values indicate power spikes that may trigger thermal or voltage regulation events.

A.2.19. CPU stability index

To evaluate the steadiness of processor utilization, the CPU stability index, csi, is defined as:

$$\mathrm{csi} = 1 - \frac{\sigma_{\mathrm{CPU}}}{\max\left(100, \mathrm{acup_{percent}}\right)}, \tag{33}$$

where σ_{CPU} is the standard deviation of CPU usage readings and $\mathrm{acup_{percent}}$ is the average CPU utilization (Eq. (15)). Values closer to one denote more uniform CPU loads.

A.2.20. Power efficiency index

The power efficiency index, pei, captures how effectively tokens are generated per watt of power:

$$\mathrm{pei} = \frac{\mathrm{tps}}{\mathrm{ap_w}}, \tag{34}$$

where tps is tokens per second (Eq. (14)). A greater pei reflects lower energy cost per token.

A.2.21. Model efficiency index

Model efficiency, mei, normalizes token throughput by peak memory usage:

$$\mathrm{mei} = \frac{\mathrm{tps}}{\mathrm{pru_{mb}}}, \tag{35}$$

with $\mathrm{pru_{mb}}$ defined in Eq. (18). Higher mei indicates more tokens generated per megabyte at peak demand.

A.2.22. Memory-to-CPU ratio

The memory-to-CPU ratio, mtcr, examines whether memory demands grow in tandem with processor load:

$$\mathrm{mtcr} = \frac{\mathrm{pru_{mb}}}{\mathrm{pcu_{percent}}}, \tag{36}$$

where $\mathrm{pcu_{percent}}$ comes from Eq. (16). Imbalanced values can indicate suboptimal resource utilization.

A.2.23. Memory-to-power ratio

To assess the influence of memory usage on energy draw, the memory-to-power ratio, mtpr, is defined as:

$$\mathrm{mtpr} = \frac{\mathrm{aru_{mb}}}{\mathrm{ap_w}}, \tag{37}$$

linking average RAM consumption to mean power usage.

A.2.24. RAM usage variation index

The RAM usage variation index, ruvi, normalizes the memory standard deviation by the average RAM usage:

$$\mathrm{ruvi} = \frac{\mathrm{msd_{mb}}}{\mathrm{aru_{mb}}}, \tag{38}$$

mirroring the definition of mvi but emphasizing variability per unit memory.

A.2.25. Power spike

Finally, the power spike metric, ps, captures the absolute difference between peak and minimum power:

$$\mathrm{ps} = \mathrm{pp_w} - \mathrm{mp_w}, \tag{39}$$

where $\mathrm{mp_w}$ is from Eq. (21). This value quantifies the energy burst magnitude during inference.

A.3. Derived LLM evaluation metrics

Building on the core and resource-aware measurements, derived metrics synthesize these values to reveal deeper aspects of an LLM's operational profile, including its responsiveness, overhead distribution, and per-token resource footprint.

A.3.1. Time per token

The time per token, tpt, quantifies the average latency incurred for generating each token. Shorter tpt values reflect more rapid token production, as expressed by

$$\mathrm{tpt} = \frac{\mathrm{total\_duration\_s}}{\mathrm{ec}}, \tag{40}$$

where $\mathrm{total\_duration\_s}$ is the overall inference time in seconds and ec is the total token count (Eq. (13)).

A.3.2. Load-to-inference ratio

To understand the relative cost of initialization versus generation, the load-to-inference ratio, ltir, is defined as

$$\mathrm{ltir} = \frac{\mathrm{ld_{ns}}}{\mathrm{ed_{ns}}}, \tag{41}$$

where $\mathrm{ld_{ns}}$ and $\mathrm{ed_{ns}}$ are the load and evaluation durations (Eqs. (10) and (12)). Larger values indicate that model startup contributes disproportionately to total latency.

A.3.3. Memory usage per token

The memory usage per token, mupt, assists in assessing memory scalability by normalizing average RAM consumption against throughput:

$$\mathrm{mupt} = \frac{\mathrm{aru_{mb}}}{\mathrm{ec}}, \tag{42}$$

where $\mathrm{aru_{mb}}$ is the average RAM usage (Eq. (17)). Lower mupt values suggest more memory-efficient token production.

A.3.4. Energy per token

Energy per token, ept, captures the joule cost of producing a single token by combining average power draw with inference duration:

$$\mathrm{ept} = \frac{\mathrm{ap_w} \times T_{\mathrm{local}}}{\mathrm{ec}}, \tag{43}$$

where $\mathrm{ap_w}$ denotes the average power (Eq. (19)) and T_{local} is the inference time in seconds. Minimizing ept is essential for energy-constrained deployments.

A.3.5. Prompt evaluation ratio

The prompt evaluation ratio, per, measures the fraction of total time consumed by prompt processing:

$$\mathrm{per} = \frac{\mathrm{ped_{ns}}}{\mathrm{td_{ns}}}, \tag{44}$$

where $\mathrm{ped_{ns}}$ is the prompt evaluation duration (Eq. (11)) and $\mathrm{td_{ns}}$ is the total duration (Eq. (9)). Values approaching unity indicate heavy preprocessing overhead relative to token generation.

A.3.6. Time per prompt evaluation

The time per prompt evaluation, denoted tppe, directly captures the overhead incurred during prompt parsing and preparation. This metric equals the prompt evaluation duration:

$$\mathrm{tppe} = \mathrm{ped_{ns}}, \tag{45}$$

where $\mathrm{ped_{ns}}$ is defined in Eq. (11). A lower tppe suggests more efficient prompt handling, which is especially important when multiple or iterative prompts are processed.

A.3.7. Prompt-to-generation overhead ratio

To compare the relative expense of prompt processing versus token generation, the prompt-to-generation overhead ratio, ptgor, is introduced:

$$\mathrm{ptgor} = \frac{\mathrm{ped_{ns}}}{\mathrm{ed_{ns}}}, \tag{46}$$

with $\mathrm{ed_{ns}}$ from Eq. (12). Higher values indicate that a larger portion of total execution time is consumed by prompt evaluation.

A.3.8. Prompt evaluation tokens per second

The prompt evaluation tokens per second, petps, reflects how many tokens the model can process during the prompt interpretation stage:

$$\mathrm{petps} = \frac{\mathrm{pec}}{\mathrm{ped_{ns}}/10^9}, \tag{47}$$

where pec is the count of tokens reread or reprocessed in the prompt. A higher petps denotes more rapid prompt throughput.

A.3.9. Evaluation latency per token

Evaluation latency per token, elpt, measures the average time the model requires to generate each token during inference:

$$\mathrm{elpt} = \frac{\mathrm{ed_{ns}}}{\mathrm{ec}}, \tag{48}$$

where ec is the total token count (Eq. (13)). Smaller elpt values indicate reduced per-token latency.

A.3.10. Token production energy efficiency

The token production energy efficiency, tpee, quantifies the number of tokens generated per joule of energy consumed:

$$\mathrm{tpee} = \frac{\mathrm{ec}}{\mathrm{ap_w} \times T_{\mathrm{local}}}, \tag{49}$$

with $\mathrm{ap_w}$ as in Eq. (19) and T_{local} the inference time in seconds. Higher tpee indicates superior energy efficiency.

A.3.11. Load-to-prompt ratio

The load-to-prompt ratio, ltpr, contrasts model initialization time against prompt processing:

$$\mathrm{ltpr} = \frac{\mathrm{ld_{ns}}}{\mathrm{ped_{ns}}}, \tag{50}$$

where $\mathrm{ld_{ns}}$ is given in Eq. (10). Values much greater than one suggest that startup overhead dominates prompt handling.

A.3.12. Prompt-to-total token ratio

The prompt-to-total token ratio, ptttr, indicates the fraction of tokens originating from the prompt versus those newly generated:

$$\mathrm{ptttr} = \frac{\mathrm{pec}}{\mathrm{ec}}, \tag{51}$$

where pec and ec are defined as above. A lower ratio implies that the model produces more novel tokens relative to prompt tokens.

A.3.13. Sustained inference factor

To capture steady-state operational efficiency, the sustained inference factor, sif, combines the proportion of generative output with energy-aware throughput:

$$\mathrm{sif} = \left(\frac{\mathrm{ec}}{\mathrm{pec} + \mathrm{ec}}\right) \times \left(\frac{\mathrm{tps}}{\mathrm{ap_w}}\right), \tag{52}$$

where tps is tokens per second (Eq. (14)). Higher sif values reflect a favorable balance between fresh token generation and power-efficient operation.

References

[1] Z. Li, X. Wu, H. Du, H. Nghiem, G. Shi, Benchmark evaluations, applications, and challenges of large vision language models: A survey, 2025, arXiv preprint arXiv:2501.02189.

[2] V. Veeramachaneni, Large language models: A comprehensive survey on architectures, applications, and challenges, Adv. Innov. Comput. Program. Lang. 7 (1) (2025) 20–39.

[3] M. Shao, A. Basit, R. Karri, M. Shafique, Survey of different large language model architectures: Trends, benchmarks, and challenges, IEEE Access (2024).

[4] J. Li, W. Lu, H. Fei, M. Luo, M. Dai, M. Xia, Y. Jin, Z. Gan, D. Qi, C. Fu, Y. Tai, A survey on benchmarks of multimodal large language models, 2024, arXiv preprint arXiv:2408.08632.

[5] Z. Li, X. Wu, H. Du, H. Nghiem, G. Shi, Benchmark evaluations, applications, and challenges of large vision language models: A survey, 2025, arXiv preprint arXiv:2501.02189.

[6] C. Xu, S. Guan, D. Greene, M. Kechadi, Benchmark data contamination of large language models: A survey, 2024, arXiv preprint arXiv:2406.04244.

[7] W.X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, A survey of large language models, 2023, arXiv preprint arXiv:2303.18223.

[8] Z. Zheng, K. Ning, Y. Wang, J. Zhang, D. Zheng, M. Ye, J. Chen, A survey of large language models for code: Evolution, benchmarking, and future trends, 2023, arXiv preprint arXiv:2311.10372.

[9] B. Li, Y. Ge, Y. Ge, G. Wang, R. Wang, R. Zhang, Y. Shan, SEED-bench: Benchmarking multimodal large language models, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2024, pp. 13299–13308.

[10] H. Zhou, F. Liu, B. Gu, X. Zou, J. Huang, J. Wu, Y. Li, S.S. Chen, P. Zhou, J. Liu, Y. Hua, A survey of large language models in medicine: Progress, application, and challenge, 2023, arXiv preprint arXiv:2311.05112.

[11] Y. Bai, J. Ying, Y. Cao, X. Lv, Y. He, X. Wang, J. Yu, K. Zeng, Y. Xiao, H. Lyu, J. Zhang, Benchmarking foundation models with language-model-as-an-examiner, Adv. Neural Inf. Process. Syst. 36 (2024).

[12] Z. Feng, W. Ma, W. Yu, L. Huang, H. Wang, Q. Chen, W. Peng, X. Feng, B. Qin, Trends in integration of knowledge and large language models: A survey and taxonomy of methods, benchmarks, and applications, 2023, arXiv preprint arXiv:2311.05876.

[13] P. Liang, R. Bommasani, T. Lee, D. Tsipras, D. Soylu, M. Yasunaga, Y. Zhang, D. Narayanan, Y. Wu, A. Kumar, B. Newman, Holistic evaluation of language models, 2022, arXiv preprint arXiv:2211.09110.

[14] Q. Xie, W. Han, X. Zhang, Y. Lai, M. Peng, A. Lopez-Lira, J. Huang, Pixiu: A large language model, instruction data and evaluation benchmark for finance, 2023, arXiv preprint arXiv:2306.05443.

[15] J. Li, X. Cheng, W.X. Zhao, J.Y. Nie, J.R. Wen, Halueval: A large-scale hallucination evaluation benchmark for large language models, 2023, arXiv preprint arXiv:2305.11747.

[16] L. Xu, A. Li, L. Zhu, H. Xue, C. Zhu, K. Zhao, H. He, X. Zhang, Q. Kang, Z. Lan, Superclue: A comprehensive chinese large language model benchmark, 2023, arXiv preprint arXiv:2307.15020.

[17] T. Shen, S. Li, Q. Tu, D. Xiong, Roleeval: A bilingual role evaluation benchmark for large language models, 2023, arXiv preprint arXiv:2312.16132.

[18] X. Liu, X. Lei, S. Wang, Y. Huang, Z. Feng, B. Wen, J. Cheng, P. Ke, Y. Xu, W.L. Tam, X. Zhang, Alignbench: Benchmarking chinese alignment of large language models, 2023, arXiv preprint arXiv:2311.18743.

[19] Y. Zhao, J. Zhang, I. Chern, S. Gao, P. Liu, J. He, Felm: Benchmarking factuality evaluation of large language models, Adv. Neural Inf. Process. Syst. 36 (2024).

[20] Q. Huang, J. Vora, P. Liang, J. Leskovec, Benchmarking large language models as ai research agents, in: NeurIPS 2023 Foundation Models for Decision Making Workshop, 2023.

[21] T. Zhang, F. Ladhak, E. Durmus, P. Liang, K. McKeown, T.B. Hashimoto, Benchmarking large language models for news summarization, Trans. Assoc. Comput. Linguist. 12 (2024) 39–57.

[22] R. Xu, Z. Wang, R.Z. Fan, P. Liu, Benchmarking benchmark leakage in large language models, 2024, arXiv preprint arXiv:2404.18824.

[23] Q. Xie, W. Han, X. Zhang, Y. Lai, M. Peng, A. Lopez-Lira, J. Huang, Pixiu: A comprehensive benchmark, instruction dataset and large language model for finance, Adv. Neural Inf. Process. Syst. 36 (2024).

[24] Y. Chen, H. Wang, S. Yan, S. Liu, Y. Li, Y. Zhao, Y. Xiao, Emotionqueen: A benchmark for evaluating empathy of large language models, 2024, arXiv preprint arXiv:2409.13359.

[25] Z. Wang, CausalBench: A comprehensive benchmark for evaluating causal reasoning capabilities of large language models, in: Proceedings of the 10th SIGHAN Workshop on Chinese Language Processing, SIGHAN-10, 2024, pp. 143–151.

[26] Z. Qiu, J. Li, S. Huang, X. Jiao, W. Zhong, I. King, Clongeval: A chinese benchmark for evaluating long-context large language models, 2024, arXiv preprint arXiv:2403.03514.

[27] A.C. Doris, D. Grandi, R. Tomich, M.F. Alam, M. Ataei, H. Cheong, F. Ahmed, Designqa: A multimodal benchmark for evaluating large language models? understanding of engineering documentation, J. Comput. Inf. Sci. Eng. 25 (2) (2025).

[28] X. Zhang, C. Li, Y. Zong, Z. Ying, L. He, X. Qiu, Evaluating the performance of large language models on gaokao benchmark, 2023, arXiv preprint arXiv:2305.12474.

[29] I. Jahan, M.T.R. Laskar, C. Peng, J.X. Huang, A comprehensive evaluation of large language models on benchmark biomedical text processing tasks, Comput. Biol. Med. 171 (2024) 108189.

[30] S. Wang, P. Wang, T. Zhou, Y. Dong, Z. Tan, J. Li, Ceb: Compositional evaluation benchmark for fairness in large language models, 2024, arXiv preprint arXiv:2407.02408.

[31] Y. Wang, Z. Yu, Z. Zeng, L. Yang, C. Wang, H. Chen, C. Jiang, R. Xie, J. Wang, X. Xie, W. Ye, Pandalm: An automatic evaluation benchmark for llm instruction tuning optimization, 2023, arXiv preprint arXiv:2306.05087.

[32] A. Berti, H. Kourani, W.M. van der Aalst, PM-LLM-benchmark: Evaluating large language models on process mining tasks, 2024, arXiv preprint arXiv:2407.13244.

[33] X. Zhou, H. Zhao, Y. Cheng, Y. Cao, G. Liang, G. Liu, W. Liu, Y. Xu, J. Zhao, Elecbench: a power dispatch evaluation benchmark for large language models, 2024, arXiv preprint arXiv:2407.05365.

[34] P. Xu, W. Shao, K. Zhang, P. Gao, S. Liu, M. Lei, F. Meng, S. Huang, Y. Qiao, P. Luo, Lvlm-ehub: A comprehensive evaluation benchmark for large vision-language models, IEEE Trans. Pattern Anal. Mach. Intell. (2024).

[35] L. Sun, Y. Han, Z. Zhao, D. Ma, Z. Shen, B. Chen, L. Chen, K. Yu, Scieval: A multi-level large language model evaluation benchmark for scientific research, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 38, no. 17, 2024, pp. 19053–19061.

[36] S. Deng, W. Xu, H. Sun, W. Liu, T. Tan, J. Liu, A. Li, J. Luan, B. Wang, R. Yan, S. Shang, Mobile-bench: An evaluation benchmark for llm-based mobile agents, 2024, arXiv preprint arXiv:2407.00993.

[37] G. Yang, C. He, J. Guo, J. Wu, Y. Ding, A. Liu, H. Qin, P. Ji, X. Liu, Llmcbench: Benchmarking large language model compression for efficient deployment, 2024, arXiv preprint arXiv:2410.21352.

[38] D. Chen, R. Chen, S. Zhang, Y. Liu, Y. Wang, H. Zhou, Q. Zhang, Y. Wan, P. Zhou, L. Sun, Mllm-as-a-judge: Assessing multimodal llm-as-a-judge with vision-language benchmark, 2024, arXiv preprint arXiv:2402.04788.

[39] L. Spangher, T. Li, W.F. Arnold, N. Masiewicki, X. Dotiwalla, R. Parusmathi, P. Grabowski, E. Ie, D. Gruhl, Project MPG: towards a generalized performance benchmark for LLM capabilities, 2024, arXiv preprint arXiv:2410.22368.

[40] G. Son, D. Yoon, J. Suk, J. Aula-Blasco, M. Aslan, V.T. Kim, S.B. Islam, J. Prats-Cristia, L. Tormo-Banuelos, S. Kim, MM-eval: A multilingual meta-evaluation benchmark for LLM-as-a-judge and reward models, 2024, arXiv preprint arXiv:2410.17578.

[41] K. Valmeekam, M. Marquez, A. Olmo, S. Sreedharan, S. Kambhampati, Planbench: An extensible benchmark for evaluating large language models on planning and reasoning about change, Adv. Neural Inf. Process. Syst. 36 (2024).

[42] J. Liu, C. Liu, P. Zhou, Q. Ye, D. Chong, K. Zhou, Y. Xie, Y. Cao, S. Wang, C. You, P.S. Yu, Llmrec: Benchmarking large language models on recommendation task, 2023, arXiv preprint arXiv:2308.12241.

[43] X. Liu, Y. Zhu, J. Gu, Y. Lan, C. Yang, Y. Qiao, Mm-safetybench: A benchmark for safety evaluation of multimodal large language models, in: European Conference on Computer Vision, Springer, Cham, 2025, pp. 386–403.

[44] Z. Chu, Q. Ai, Y. Tu, H. Li, Y. Liu, Pre: A peer review based large language model evaluator, 2024, arXiv preprint arXiv:2401.15641.

[45] J. Yu, X. Wang, S. Tu, S. Cao, D. Zhang-Li, X. Lv, H. Peng, Z. Yao, X. Zhang, H. Li, C. Li, Kola: Carefully benchmarking world knowledge of large language models, 2023, arXiv preprint arXiv:2306.09296.

[46] Ollama, https://github.com/ollama/ollama/blob/main/docs/api.md. (Accessed 11 January 2025).