BENCH360—BENCHMARKING LOCAL LLM INFERENCE FROM 360°

Anonymous authors
Paper under double-blind review

ABSTRACT

Running large language models (LLMs) locally is becoming increasingly common. While the growing availability of small open-source models and inference engines has lowered the entry barrier, users now face an overwhelming number of configuration choices. Identifying an optimal configuration—balancing functional and non-functional requirements— requires substantial manual effort. While several benchmarks target LLM inference, they are designed for narrow evaluation goals and not user-focused. They fail to integrate relevant system and task-specific metrics into a unified, easy-to-use benchmark that supports multiple inference engines, usage scenarios, and quantization levels. To address this gap, we present *Bench360—Benchmarking Local LLM Inference from 360°*. Bench360 allows users to easily define their own custom tasks along with datasets and relevant task-specific metrics and then automatically benchmarks selected LLMs, inference engines, and quantization levels across different usage scenarios (single stream, batch & server). Bench360 tracks a wide range of metrics, including (1) *system metrics*—such as *Computing Performance* (e.g., latency, throughput), *Resource Usage* (e.g., energy per query), and *Deployment* (e.g., cold start time)—and (2) *task-specific metrics* such as ROUGE, F1 score or accuracy. We demonstrate Bench360 on four common LLM tasks—General Knowledge & Reasoning, QA, Summarization and Text-to-SQL—across three hardware platforms and four state of the art inference engines. Our results reveal several interesting trade-offs between task performance and system-level efficiency, highlighting the differences in inference engines and models. Most importantly, there is no single best setup for local inference, which strongly motivates the need for a framework such as Bench360.

1 INTRODUCTION

Recent advances in quantization techniques (e.g., BitsAndBytes Dettmers et al. (2022; 2023) AutoAWQ Lin et al. (2023), GPTQ Frantar et al. (2022)), lightweight model architectures (e.g., Llama Touvron et al. (2023), Gemma Team et al. (2024), Qwen Yang et al. (2024) and Mistral Jiang et al. (2023)), and inference frameworks (e.g., `HuggingFace TGI` Face (2025), `vLLM` Kwon et al. (2023a), `SGLang` Zheng et al. (2024b), LMDeploy Contributors (2023); Zhang et al. (2025)) have made local LLM deployments more feasible for a broader user base. This growing user base uses LLMs for *various tasks* across *different deployment scenarios*. Besides task-specific functional quality metrics (e.g., accuracy, F1), their deployments have varied non-functional requirements related to system performance (e.g., latency/throughput) and resource usage (e.g. energy). All this constitutes a complex *LLM inference design space* in which users deploying local models face the critical challenge: *Which combination of model, quantization, and inference framework offers the best trade-off between computational cost, energy consumption, and output quality for my specific task and deployment scenario?*

While users can manually implement specific configurations and iteratively find a working solution through trial and error, this requires considerable time and effort and can still miss the best configuration. Users need to monitor relevant system performance and resource metrics alongside task-specific metrics like accuracy. What is missing is a benchmarking framework that allows them to: (1) select a set of potentially promising LLMs; (2) define the task with relevant task-specific metrics; (3) define their specific workload scenario; and (4) choose inference engines to evaluate.

The framework should then provide standardized evaluation across these configurations and deliver comprehensive performance data to help users identify suitable configurations for their specific requirements. This is exactly what we propose with *Bench360—Benchmarking Local LLM Inference from 360°*: `https://anonymous.4open.science/r/bench360-9122`. Our framework systematically evaluates configurations across models, inference engines, quantization schemes, and deployment scenarios, providing users with actionable insights to make informed deployment decisions rather than relying on costly trial-and-error approaches.

While many benchmarks have been presented to evaluate large language models—such as MLPerf Inference Reddi et al. (2019), Holistic Evaluation of Language Models (HELM) Liang et al. (2023), and the HuggingFace Open LLM Leaderboard Beeching et al. (2023)—most of these focus on cloud-scale deployment, model accuracy, or leaderboard-style comparison. They often lack support for key deployment metrics such as energy efficiency, cold start and latency—all of which are critical for realistic on-premise deployment in constrained environments. Furthermore, many existing benchmarks are tightly coupled to specific APIs or evaluation scenarios, limiting their adaptability (see Section 2 for a detailed discussion).

In contrast, we introduce a benchmarking framework tailored to the underexplored but increasingly important setting of local LLM deployment for specific tasks, emphasizing supporting practical deployment issues. Overall, Bench360 provides the following key features:

- **System and Task Metrics.** Bench360 combines system and task-specific metrics in one single framework. We integrate not only common efficiency metrics, including latency, throughput, memory usage, cold-start behavior, concurrency resilience and energy consumption, but also task-specific quality metrics (e.g., accuracy, F1, ROUGE) that users can define freely. *This enables the selection of a configuration based on individual trade-offs.*
- **Multiple Inference Frameworks and Quantizations.** Bench360 supports multiple LLM model families, quantization schemes, and inference frameworks, enabling a direct comparison across popular frameworks such as `HuggingFace TGI`, `vLLM`, `SGLang`, or `LMDeploy`. *This enables users to find the best combination of model architecture, quantization and inference framework.*
- **Custom Task and Inference Scenario Definition.** Most importantly, Bench360 comes with plug-and-play support for new tasks and model configurations, designed to enable reproducible experiments and continuous benchmarking. *This allows users to run Bench360 not only for pre-defined tasks, but lets them easily define and benchmark their own.*

2 Related Work

The widespread adoption of LLMs has led to a growing interest in benchmarking their inference performance. LLMeBench Dalvi et al. (2023) enables the evaluation of the accuracy of LLMs on different NLP tasks, but does not provide any efficiency metrics. The Open LLM Leaderboard Beeching et al. (2023) by Hugging Face focuses on evaluating the task-specific accuracy of models, but only reports the estimated carbon emissions during their evaluation. The Edge LLM Leaderboard Arnav Chavan (2024) further includes system metrics such as time-to-first-token (TTFT) and inter-token-latency. LLMPerf Contributors (2024) evaluates the output quality via correctness and system metrics via a load test, but does not evaluate local deployments, only LLM APIs. Vidur-Bench Agrawal et al. (2024) is an LLM benchmark for evaluating the performance of LLMs through workload patterns (tasks), scheduling, batching, and routing policies, and serving frameworks. Vidur-Bench is part of an LLM simulation framework, and new models require several steps that make it not easy to evaluate and compare multiple LLMs. *Notably, all above benchmarks do not measure the energy consumption of LLMs.* MLPerf Mattson et al. (2020), a joint effort between academia and industry, develops a benchmark suite to evaluate the training and inference efficiency, as well as the quality, safety and energy of ML systems. MLPerf includes various benchmarks for training and inference in different domains (e.g, language, computer vision, and recommender systems). Reference implementations are provided for multiple models in four different scenarios (single stream, multiple stream, server, and offline). Unlike the previous benchmarks, MLPerf considers energy consumption through the MLPerf Power workgroup that measures energy consumption while running the MLPerf benchmarks through the use of a physical power meter—currently, software measurements are not accepted for submission Tschand et al. (2024). LLM-Inference-Bench Chitty-Venkata et al. (2024) is designed to

evaluate the inference performance of LLMs in the Llama family and derivatives, such as Gemma, and Qwen, across multiple frameworks and hardware. However, LLM-Inference-Bench is task-agnostic, using only perplexity to evaluate the quality of LLMs, which is not a sufficient metric to approximate task-specific metrics Nimah et al. (2023). Last, ML.ENERGY Chung et al. (2025) addresses the automated energy optimization for large-scale LLM inference, focusing on Pareto-frontier-based configuration tuning for high-end data center GPUs. In contrast to these system-oriented benchmarks, Bench360 streamlines the evaluation of LLMs across tasks and scenarios on a given deployment environment. Table 1 summarizes the differences of Bench360 compared to existing works. *Compared to all existing works we cover the entire local deployment stack including a wide range of system metrics and customizability of task, quantization and scenario.*

Table 1: Overview of core capabilities in LLM benchmarks compared to Bench360

Benchmark	Local	Supported System Metrics			Custom Benchmark Definition		
		Performance	Resources	Deployment	Tasks	Quantization	Scenarios
Open LLM Lead. Beeching et al. (2023)	✗	✗	CO2 emissions	✗	✗	✗	✗
Edge LLM Lead. Arnav Chavan (2024)	✓	Latency	Model size	✗	✗	✗	✗
LLMPerf Contributors (2024)	✗	Latency	Energy	✗	✗	✗	✗
Vidur-Bench Agrawal et al. (2024)	✓	Latency, TPS, Memory usage	MFU, KV Cache Util.	✗	✗	✓	✗
MLPerf Inference Tschand et al. (2024)	✓	Latency, TPS	Energy	✗	✗	✓	✓
LLM-Inference-Bench Chitty-Venkata et al. (2024)	✓	Latency	Energy	✗	✓	✓	✗
ML.ENERGY Chung et al. (2025)	✓	Latency	Energy	✗	✗	✗	✗
Bench360 (ours)	✓	Latency, Peak Memory usage, GPU util.	Energy, Cost per query, Model size, Overhead	Cold start time, OOM resil., Multi-user-concurrency	✓	✓	✓

3 BENCH360—BENCHMARKING LOCAL LLMS FROM 360°

Bench360 comprises four components: (1) a **task engine** that standardizes input/output for tasks such as summarization, QA, and Text-to-SQL; (2) a **workload controller** that simulates usage scenarios (single-stream, batch, server) with configurable query patterns; (3) a **backend abstraction layer** for executing models via different inference engines (currently, HuggingFace TGI, vLLM, SGLang and LMDeploy); and (4) a **metrics collector** that captures system metrics such as latency, throughput, memory usage, energy consumption, and task-specific metrics.

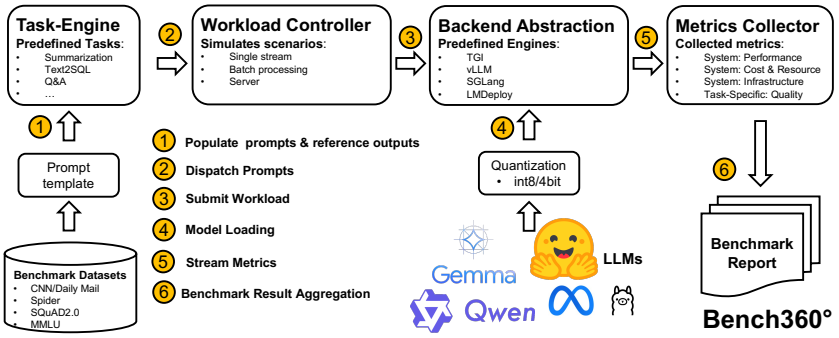


Figure 1: **Bench360.** Datasets plus a prompt template yield task-specific prompts that enter the *Task-Engine* (1) and pass to the *Workload Controller* (2), which simulates single, batch, or server traffic before handing the workload to the *Backend Abstraction* (3). The backend loads a quantized LLM and executes inference on the selected engines (4); generated text and runtime traces stream into the *Metrics Collector* (5), which aggregates system and task metrics into a final benchmark report (6).

Bench360 provides a command-line interface and configuration-based workflow. Here, users configure the benchmark via a YAML configuration file (Table 2) without modifying code. This file specifies the task type, dataset, evaluation metrics, model configuration, backend engine, and quantization format. For advanced use, developers can extend the framework by implementing new tasks, inference engines, or workload generators using the modular API defined in the benchmark core. Models can be retrieved directly from the Hugging Face Hub in the required format, including quantized variants, provided that the chosen inference engine supports them.

Table 2: YAML configuration fields with descriptions and possible values.

Field	Description	Possible Values
`hf_model`	Hugging Face model identifier or path to the LLM	`[google/gemma-2-9b-it, ...]`
`backend`	Inference engine for model execution	`[tgi, vllm, sglang, lmdeploy]`
`quantization`	Precision format for model weights	`[fp16, int8, 4bit, awq, gptq]`
`task`	Evaluation task type	`[mmlu, summarization, qa, sql]`
`scenario`	Execution mode for inference workload	`[single, batch, server]`
`samples`	Number of examples for batch/single scenarios	`[64, 128, 256, ...]`
`batch_size`	Batch size for batch processing scenario	`[8, 16, 32, ...]`
`run_time_s`	Duration in seconds for server scenario	`[300, 600, 1200, ...]`
`concurrent_users`	Number of simultaneous users for server scenario	`[1, 8, 16, 32, ...]`
`requests_per_user_per_min`	Request rate per user for server scenario	`[8, 16, 32, ...]`

3.1 Task Engine

The task engine operates through a standardized plugin architecture where each task is linked to a dataset via a path or identifier (e.g., `mmlu`, `squad_v2`, `cnn_dailymail`, or custom local files). Each task implements the `BaseTask` interface with two core methods: `generate_prompts()` which creates test prompts and extracts reference answers from the dataset, and `quality_metrics()` which computes task-specific evaluation scores by comparing generated outputs with reference answers. The benchmark automatically handles input formatting and output parsing for each task type. Tasks are dynamically loaded and instantiated based on configuration parameters, allowing the system to seamlessly switch between different evaluation scenarios. The task engine abstracts away dataset-specific preprocessing and metric computation, providing a unified interface that integrates with the broader benchmarking framework's inference engines and deployment scenarios.

3.2 Workload Controller

Inspired by MLPerf Mattson et al. (2020), the controller emulates three serving scenarios—*single-stream*, *batch*, and *server*—each implemented by a unified event loop that advances in fixed time steps. **Single-stream.** Exactly one prompt is kept ready at all times. Whenever the backend becomes idle, that prompt is sent, the loop blocks until the answer is returned, and the next prompt is loaded. No queue is formed. **Batch.** All prompts are available before the timer starts. The controller accumulates a batch of size b and submits it in one call. This repeats until all samples are processed. **Server.** The server scenario simulates concurrent users making requests to the inference backend. Each user generates requests independently following a Poisson process with a predefined rate of queries per minute. The benchmark uses a thread pool to dispatch requests immediately to the backend as they arrive, there is no artificial queuing or request batching imposed by the benchmark itself. Any request queuing, batching, or scheduling behavior observed during execution is entirely determined by the backend's internal implementation and capacity constraints.

3.3 Backend Abstraction

The **Backend Abstraction** layer provides a unified interface for multiple inference engines including Hugging Face TGI, vLLM, SGLang and LMDeploy. Each backend is containerized using Docker to ensure consistent deployment and isolation. At startup, the system launches the appropriate container with the requested model, loading from local storage or automatically downloading from Hugging Face Hub. The abstraction layer translates unified configuration parameters into each engine's native API format and dispatches requests through standardized OpenAI-compatible endpoints. This design allows seamless comparison between different engines and quantization schemes without configuration changes, while new runtimes can be integrated by implementing the common backend interface.

3.4 System Metrics Collector

We categorize system metrics into *Performance*, *Resource*, and *Deployment* (Table 3). **Performance.** We adopt the full NVIDIA NIM metrics suite NVIDIA Corporation (2025): We measure latency at different granularities including time-to-first-token (TTFT), time per output token (TPOT), and generation latency (GL). Throughput is tracked in tokens per second (TPS) and sentences per second

(SPS). These metrics are recorded during generation using consistent timing hooks across all backends. **Resources.** We track GPU utilization, memory footprint, CPU utilization, and RAM usage of the inference process using NVML and psutil telemetry. Energy consumption is measured via NVML with accuracy within $\pm 5W$ Kasichayanula et al. (2012). Model size and memory overhead are computed from disk footprint and runtime memory deltas. Energy efficiency is measured as energy per token and energy per sentence.

Deployment. We evaluate comprehensive deployment metrics including container startup time, model loading time, and cold-start latency. For server scenarios, we additionally measure queue time (request submission to processing delay), wait time (scheduling overhead), and end-to-end latency to capture real-world deployment behavior under concurrent load.

Table 3: System Evaluation Metrics Overview

Category	Aspect	Metric	Measurement Method/Details
Performance	Latency	TTFT, TPOT, GL	Measured during inference with timing hooks
	Throughput	TPS, SPS	Tokens and sentences per second
	Server Latency	Queue time, wait time, E2E latency	Thread pool and scheduling delays
Resources	GPU Memory	Peak/average GPU memory (MB)	NVML memory monitoring
	GPU Utilization	Peak/average GPU utilization (%)	NVML utilization rates
	CPU/RAM Usage	Peak/average CPU utilization, RAM (MB)	psutil process monitoring
	Power	Peak/average power (W), total energy (Wh)	NVML power telemetry
	Energy Efficiency	Energy per token/sentence (J)	Power × generation time
	Memory Overhead	Runtime memory beyond model size	GPU memory delta
Deployment	Cold Start	Container startup, TTFT, total coldstart	Timer from launch to first token
	Model Size	Disk footprint (MB)	Safetensors/checkpoint file size

4 Experimental Evaluation

While Bench360 is a general-purpose LLM inference benchmark, we rely on a set of research questions to guide us in our investigation and to show the unique benefits of our framework. Specifically, for different LLM model sizes/families, for different hardware platforms and four tasks, we investigate the following questions.

[RQ1] *Memory-Constrained Deployment*: Given a strict 24GB VRAM budget (typical of mid-tier GPUs), how should memory be allocated most effectively? Is it better to run a smaller full-precision model (e.g., 7B FP16) or a larger quantized counterpart (e.g., 14B INT8, 24–32B INT4) in terms of overall effectiveness (quality) and efficiency (latency, energy per token)?

[RQ2] *Scenario-Focused Serving*: Across the three serving scenarios — single-stream responsiveness, offline batch throughput, and multi-user concurrency — which inference engine (vLLM, SGLang, LMDeploy, TGI) offers the best performance on a given GPU platform?

[RQ3] *System Efficiency*: Do inference engines differ in their energy consumption, and which engine–GPU combinations achieve the best efficiency relative to GPU utilization?

4.1 Experimental Setup

Tasks, datasets, and metrics. We evaluate four tasks that reflect common LLM applications. Multitask Language Understanding covers reasoning, knowledge retrieval, and comprehension using the MMLU benchmark dataset Hendrycks et al. (2020), which is evaluated using the accuracy. Extractive Question Answering is configured with SQuAD2.0 Rajpurkar et al. (2018), where we measure F1 score. Summarization is evaluated on the CNN/DailyMail dataset See et al. (2017), with ROUGE-L as the main metric. Text-to-SQL uses the Spider dataset Yu et al. (2018) with Execution Accuracy (EA) as the primary metric Fürst et al. (2025); Sivasubramaniam et al. (2024); Yu et al. (2018).

LLMs, inference engines, and hardware. We evaluate four open-weight LLM families (LLaMA-3, Gemma-2, Mistral, Qwen2.5) within a 24GB VRAM constraint to reflect realistic local deployments. For *memory-constrained deployment*, we compare smaller FP16 models against larger GPTQ-quantizedFrantar et al. (2023) variants from the same families: Gemma-2 (9B FP16 vs. 27B INT4), Mistral (7B FP16 vs. 24B INT4), and Qwen2.5 (7B FP16, 14B INT8, 32B INT4) using vLLM

across L4, A10, and A30 GPUs. For *scenario-focused serving*, we benchmark three representative mid-size models (Mistral-7B-Instruct-v0.3, Qwen2.5-7B-Instruct, LLaMA-3.1-8B-Instruct) across four inference engines (vLLM, SGLang, TGI, LMDeploy) on all GPU tiers. For *system efficiency*, we use the same three models and four engines across all three GPUs to isolate performance differences (Tables7, 8).

4.2 Memory-Constrained Deployment

A core deployment question is how to allocate a fixed GPU memory budget. Quantization offers one solution, as it enables larger models to fit into limited memory, but it typically comes at the cost of quality degradation Li et al. (2024). What remains under investigated is whether a larger quantized model can still outperform a smaller FP16 counterpart under the same memory constraints. To answer this, we compare, within each model family, the largest FP16 model that fits within 24 GB against the largest quantized (INT8/INT4) variant. Our evaluation spans tasks in general knowledge and reasoning (MMLU), extractive QA (SQuAD v2), summarization (CNN/DailyMail), and Text-to-SQL (Spider).

MMLU (General Knowledge & Reasoning)								
Model	**Quant**	**Accuracy**	ΔQ (%)	**TPOT (ms)**	ΔTPOT (%)	**J/Token**	ΔE (%)	**Size (GB)**
Gemma-2-it-9B	FP16	0.714	0.000	47.649	0.000	4.646	0.000	**17.627**
Gemma-2-it-27B	INT4	**0.742**	**3.854**	**125.494**	**163.373**	**12.705**	**173.477**	17.418
Mistral-7B-Instruct-v0.3	FP16	0.574	0.000	89.552	0.000	9.031	0.000	**13.825**
Mistral-Small-24B-Instruct	INT4	0.574	**0.043**	**110.488**	**18.949**	**10.443**	**13.520**	13.492
Qwen2.5-7B-Instruct	FP16	0.688	0.000	189.698	0.000	17.863	0.000	14.526
Qwen2.5-14B-Instruct	INT8	0.769	11.819	266.190	40.323	26.135	46.306	15.875
Qwen2.5-32B-Instruct	INT4	**0.794**	**15.432**	**336.934**	**77.616**	**32.926**	**84.318**	**18.448**
Squad v2 (Extractive QA)								
Model	**Quant**	**F1**	ΔQ (%)	**TPOT (ms)**	ΔTPOT (%)	**J/Token**	ΔE (%)	**Size (GB)**
Gemma-2-it-9B	FP16	0.903	0.000	36.47	0.000	4.121	0.000	**17.627**
Gemma-2-it-27B	INT4	**0.907**	**0.498**	**78.08**	**114.108**	**9.029**	**119.104**	17.418
Mistral-7B-Instruct-v0.3	FP16	0.847	0.000	35.36	0.000	4.229	0.000	**13.825**
Mistral-Small-24B-Instruct	INT4	**0.916**	**8.120**	**58.79**	**66.261**	**6.818**	**61.219**	13.492
Qwen2.5-7B-Instruct	FP16	0.871	0.000	57.39	0.000	6.321	0.000	14.526
Qwen2.5-14B-Instruct	INT8	0.885	1.572	98.52	71.662	11.242	77.831	15.875
Qwen2.5-32B-Instruct	INT4	**0.889**	**2.052**	**128.17**	**123.319**	**14.341**	**126.866**	**18.448**
CNN/DailyMail (Summarization)								
Model	**Quant**	**ROUGE-L**	ΔQ (%)	**TPOT (ms)**	ΔTPOT (%)	**J/Token**	ΔE (%)	**Size (GB)**
Gemma-2-it-9B	FP16	0.380	0.000	16.31	0.000	1.749	0.000	**17.627**
Gemma-2-it-27B	INT4	**0.395**	**3.771**	**18.21**	**11.621**	**2.087**	**19.320**	17.418
Mistral-7B-Instruct-v0.3	FP16	0.384	0.000	6.97	0.000	0.814	0.000	**13.825**
Mistral-Small-24B-Instruct	INT4	**0.396**	**3.043**	**14.92**	**113.934**	**1.750**	**114.908**	13.492
Qwen2.5-7B-Instruct	FP16	0.231	0.000	7.17	0.000	0.823	0.000	14.526
Qwen2.5-14B-Instruct	INT8	0.370	60.538	11.66	62.452	1.346	63.637	15.875
Qwen2.5-32B-Instruct	INT4	**0.378**	**63.617**	**32.47**	**352.594**	**3.682**	**347.463**	**18.448**
Spider (Text2SQL)								
Model	**Quant**	**Exec. Acc.**	ΔQ (%)	**TPOT (ms)**	ΔTPOT (%)	**J/Token**	ΔE (%)	**Size (GB)**
Gemma-2-it-9B	FP16	0.616	0.000	14.00	0.000	1.606	0.000	**17.627**
Gemma-2-it-27B	INT4	**0.657**	**6.769**	**20.01**	**39.564**	**2.297**	**43.007**	17.418
Mistral-7B-Instruct-v0.3	FP16	0.335	0.000	22.77	0.000	2.778	0.000	**13.825**
Mistral-Small-24B-Instruct	INT4	**0.528**	**57.831**	**26.18**	**14.982**	**3.024**	**8.863**	13.492
Qwen2.5-7B-Instruct	FP16	0.565	0.000	8.72	0.000	0.958	0.000	14.526
Qwen2.5-14B-Instruct	INT8	0.550	-2.619	21.08	141.746	2.376	147.972	15.875
Qwen2.5-32B-Instruct	INT4	**0.681**	**20.714**	**24.38**	**179.615**	**2.752**	**187.159**	**18.448**

Table 4: Models are grouped by task and family. Within each family, the highest value in every numeric column is highlighted in bold. All results averaged across L4, A10, and A30 GPUs using vLLM in batch-serving mode.

Table 4 and Fig. 2 reveal three consistent patterns across tasks. (i) Quantization enables the use of large models but the higher number of operations per token still requires more computing. INT8/INT4 allows 2–4$\times$ larger models within a 24GB budget and often yields higher scores, yet TPOT and energy can rise by more than $+350\%$, especially for Qwen2.5-32B INT4, impacted by the higher

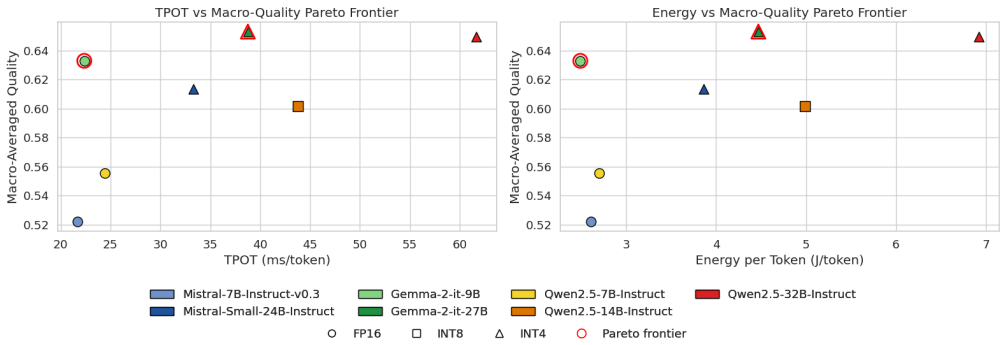


Figure 2: Latency–quality and energy–quality Pareto frontiers under a 24 GB VRAM budget. Gemma-2 variants consistently dominate both frontiers, while Qwen2.5 achieves the highest task quality at disproportionate system cost.

number of operations needed. The roofline model shows that memory and computation are the two main bottlenecks in LLM inference Yuan et al. (2024); while quantization has been shown to help in reducing memory and bandwidth bottlenecks in LLM inference Argerich & Patiño-Martínez (2024), computation is not affected by quantization, thus its benefit is limited. Other techniques such as pruning could help to reduce the computation bottleneck Chitty-Venkata et al. (2023); Park et al. (2024). (ii) Efficiency is family-dependent. Gemma-2 achieves the most balanced trade-off: its 9B FP16 baseline already sits near the Pareto frontiers, and the 27B INT4 variant improves accuracy with moderate overheads. In contrast, Mistral-24B INT4 achieves large quality gains (up to $+57.8\%$ in SQL) but with notable latency and energy penalties, while Qwen2.5 maximizes accuracy at the steepest cost in both TPOT and J/Token. (iii) System cost depends more on architecture than on size alone. Smaller FP16 baselines like Gemma-9B and Mistral-7B remain more energy-efficient than their quantized large-family counterparts, indicating that activation compute and memory bottlenecks dominate runtime, even with compressed weights and activations. Only the Gemma-2 series consistently appears on both the latency–quality and energy–quality Pareto frontiers, showing that architectural design is as critical as model scale.

4.2.1 Answering RQ1

Quantized large models can outperform smaller FP16 baselines within a fixed VRAM budget, but the trade-offs vary by family. Gemma-9B FP16 is best for latency-sensitive deployments, combining strong accuracy with the lowest TPOT and J/Token in its block. Gemma-27B INT4 provides the most balanced upgrade path, lifting accuracy without catastrophic overheads. Mistral-24B INT4 and Qwen2.5-32B INT4 achieve the highest task quality improvements (Table 4) but at the steepest system cost, making them suitable for batch-serving or offline workloads where throughput matters more than latency. Overall, optimal deployment depends on jointly considering quantization level, architecture, and task requirements; *choosing which family to scale often matters more than simply scaling further*.

4.3 Scenario–Focused Serving

A core serving question is how to provision an inference stack for a fixed GPU type and inference scenario. We study whether inference frameworks deliver the responsiveness, throughput, and energy efficiency required by the three dominant deployment regimes—*single-stream*, *offline batch*, and *multi-user concurrency*—and how these trade-offs vary across hardware.

4.3.1 Single-Stream Latency

Figure 3 summarises startup time, TTFT, and TPOT under single-stream conditions. LMDeploy exhibits the strongest end-to-end responsiveness overall: it starts fastest, produces the first token earliest, and sustains the lowest TPOT across devices. On A10 and L4, startup completes in roughly 15 s and 40 s, respectively, while A30 also benefits from markedly shorter cold-start delays than

competing engines. By contrast, TGI shows protracted and highly variable startup, exceeding 260 s on A30 with wide confidence intervals, which undermines interactive use. vLLM and SGLang start more quickly than TGI but do not match LMDeploy's cold-start profile.

First-token experience follows a similar pattern. LMDeploy and TGI obtain the lowest TTFT on all three GPUs, with LMDeploy reaching about 45ms on A30 and maintaining tight run-to-run variability. vLLM and SGLang produce competitive but consistently higher TTFTs, especially on L4 where SGLang peaks near 160ms. Once generation is underway, TPOT is lowest on A30 for all engines due to higher memory bandwidth. LMDeploy again leads, recording ∼0.09 s/token on L4 and ∼0.095 s/token on A30, with vLLM and SGLang following closely and TGI lagging at both higher TPOT and higher variance.

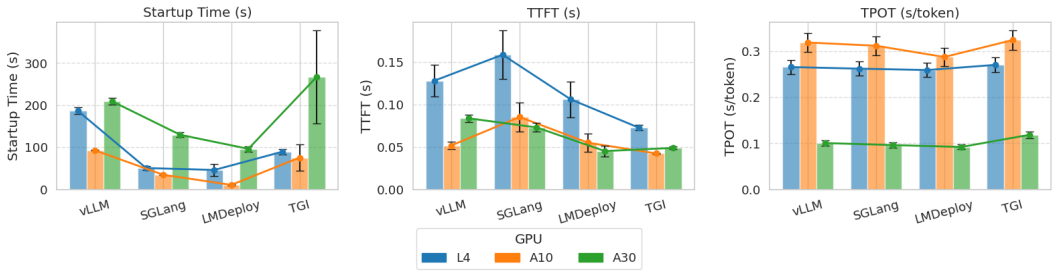


Figure 3: **Single-stream responsiveness.** Startup time, TTFT, and TPOT averaged over all tasks/models on L4, A10, and A30. Error bars indicate 95% confidence intervals.

4.3.2 Batch Throughput

We scale the throughput evaluation under increasing batch sizes $B \in \{16, 32, 64, 128\}$. Table 5 reports offline throughput scaling as batch size increases. All engines benefit from larger batches, but the winner depends on the GPU tier. On mid-range hardware (L4, A10), **SGLang** delivers the highest TPS across all batch sizes, reaching 648.5 TPS on A10 at B=128 and consistently outpacing vLLM, LMDeploy, and TGI. This suggests that SGLang's batching and kernel execution are particularly effective under bandwidth and compute constraints. On the high-end A30, **vLLM** takes the lead, peaking at 964.4 TPS for B=128, which points to runtime optimisations that better exploit tensor cores and high-bandwidth memory. LMDeploy and TGI scale less efficiently, especially on L4 and A10 where their TPS often remains below 200 at the largest batch sizes. Since all experiments use FP16 variants of Qwen2.5-7B, LLaMA-3.1-8B, and Mistral-7B, the observed differences chiefly reflect engine-level scheduling and execution efficiency rather than numerical compression.

GPU	Batch Size	vLLM	SGLang	LMDeploy	TGI
L4	16	133.0 ± 5.6	**140.7 ± 6.1**	100.4 ± 4.1	92.1 ± 3.5
	32	190.6 ± 8.8	**228.4 ± 11.1**	137.4 ± 6.1	116.5 ± 4.8
	64	304.8 ± 15.2	**364.3 ± 19.8**	162.5 ± 8.0	135.3 ± 5.9
	128	401.9 ± 21.0	**513.2 ± 29.4**	198.6 ± 11.5	166.5 ± 8.8
A10	16	155.3 ± 7.2	**173.9 ± 8.6**	112.3 ± 4.7	96.2 ± 3.7
	32	253.1 ± 12.9	**296.2 ± 16.3**	142.8 ± 6.2	120.5 ± 4.7
	64	377.8 ± 21.4	**463.0 ± 28.2**	162.3 ± 7.6	139.6 ± 5.6
	128	497.8 ± 29.6	**648.5 ± 41.4**	176.5 ± 8.6	149.0 ± 6.1
A30	16	**318.6 ± 12.7**	218.5 ± 6.7	238.6 ± 9.5	173.2 ± 6.0
	32	**442.6 ± 19.5**	362.0 ± 12.2	312.2 ± 13.6	228.2 ± 8.4
	64	**733.3 ± 34.8**	574.4 ± 23.4	353.0 ± 16.6	246.1 ± 9.0
	128	**964.4 ± 47.9**	917.6 ± 43.6	378.2 ± 18.6	263.0 ± 9.8

Table 5: **Offline batch throughput** (tokens/s ± 95% CI) across batch sizes and GPUs.

4.3.3 Server Concurrency

We now turn to multi-user serving with short-to-medium prompts and outputs (320 tokens prompts and outputs 32 tokens output in average). For concurrency we use $C \in \{8, 16, 32, 64\}$ simulated

threads at 12 rpm with Poisson arrivals and QoS (Quality of Service) thresholds of 2 s (TTFT) and 6 s (E2E). Throughput increases with concurrency until each engine-GPU pair reaches its saturation point, at which latency grows rapidly. Figure 5 (appendix) shows vLLM continues to scale up to C=64 on A30, while keeping TTFT and E2E close to their targets, whereas on A10 and L4, saturation typically occurs between C=16 and C=32 due to tighter resource budgets. Across devices, vLLM exhibits the most robust queueing and scheduling behaviour: it consistently satisfies QoS up to C=32 and remains usable at C=64 without severe tail latencies. SGLang performs well at light to moderate concurrency but breaches the TTFT threshold earlier on L4 as load increases. LMDeploy and TGI saturate sooner, with rising TTFT and E2E beyond C=16 in many settings, indicating less effective request interleaving and cache management. For chat backends and shared APIs where predictable latency under load is paramount, these results favour vLLM.

4.3.4 Answering RQ2

These results underline that *the optimal inference engine strongly depends on the serving scenario*. LMDeploy's short startup and low TTFT make it the most suitable choice when single-request responsiveness and frequent cold starts dominate the user experience, such as in CLI tools, on-demand inference, debugging loops, and serverless or edge deployments. SGLang shows clear advantages in offline batch settings on mid-tier GPUs (L4, A10), whereas vLLM dominates on upper-mid hardware (A30), where its runtime optimizations fully exploit memory bandwidth and tensor-core throughput. In multi-user concurrency, vLLM emerges as the most robust engine, consistently maintaining QoS under high load, while SGLang remains a solid second choice. By contrast, LMDeploy and TGI saturate early, making them less suitable for shared backends. Taken together, these findings imply that engine choice should be aligned with the intended deployment pattern: LMDeploy for cold-start responsiveness, SGLang for batch workloads on constrained devices, and vLLM for scalable multi-user serving, especially on high-bandwidth GPUs.

4.4 System Efficiency

Figure 4 aggregates two central efficiency metrics— energy consumption and GPU utilization—across inference engines and GPUs. Results are averaged over three representative models, four tasks, and all three serving regimes to minimise model, task or serving-specific variance.

The results show pronounced efficiency differences between engines. Energy per token and GPU utilization are not strongly correlated. vLLM consistently stands out as the most energy-efficient engine across GPUs, even though its average utilization is not always the highest. This suggests that its batching and scheduling strategies keep the GPU active only as long as necessary, thereby reducing wasted energy. In contrast, other engines may drive higher utilization but with less effective scheduling, leading to longer active periods and greater energy consumption. Hardware characteristics further amplify these effects: the A30, with its larger memory bandwidth and compute capacity, consistently reduces energy per token compared to the L4 and A10, widening the gap between efficient and less efficient engines.

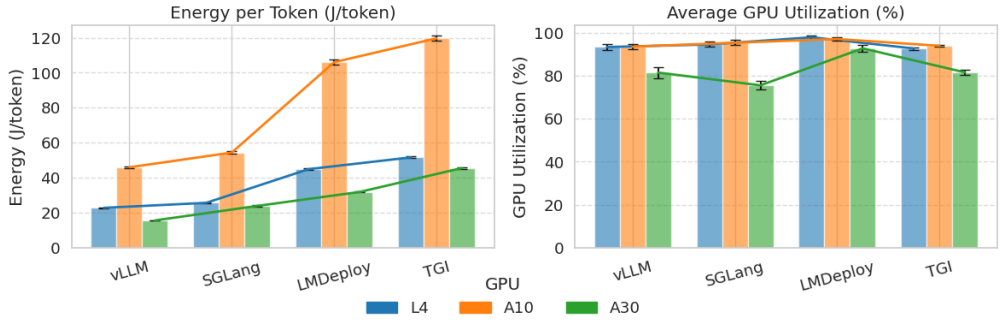


Figure 4: System-level runtime metrics across inference engines and GPUs (L4, A10, A30). Bars show GPU utilization and energy consumption per token, with 95% confidence intervals.

4.4.1 ANSWERING RQ3

Efficiency depends as much on engine scheduling as on hardware. High GPU utilization does not guarantee low energy cost: engines may keep devices busy without using resources effectively. vLLM shows that careful batching and scheduling can deliver the best energy efficiency even at moderate utilization. For practitioners, this means engine choice should consider how well GPU activity is converted into useful work. Hardware upgrades (e.g., L4 to A30) further reduce energy per token, but only when paired with engines that can exploit the extra bandwidth and compute. Overall, vLLM provides the most balanced trade-off between energy efficiency and utilization across GPU tiers.

5 LIMITATIONS

While Bench360 is the first LLM inference benchmark that enables a practical, deployment-focused evaluation of system and task metrics, we note several limitations, some of which provide the potential for further research:

Different quantization techniques. In this study we focused on GPTQ Frantar et al. (2023), but Bench360 is designed to support a wider range of quantization methods. Extending coverage to additional techniques will provide even richer insights once engine support broadens.

Larger models and wider hardware selection. Our evaluation was conducted on mid-tier GPUs with 24GB memory, a realistic setup for smaller labs, organizations, and individual practitioners. This focus ensures strong relevance to typical local deployments. Future work will build on this foundation by scaling to larger models and a broader set of hardware.

Inference Engine Coverage. We selected four inference engines (vLLM, SGLang, TGI, LMDeploy) that enable fair comparison by supporting identical model formats, Hugging Face Hub integration, and standardized APIs. Other popular engines like llama.cpp/Ollama require custom model formats (GGUF), and tools like ExLlamaV2 or TensorRT-LLM have specialized requirements that prevent controlled comparison. Our selection prioritizes engines where performance differences reflect genuine optimization rather than format conversion overhead or abstraction layers. The inference engine landscape evolves rapidly, and our modular framework can accommodate additional engines as they adopt standardized interfaces.

CPU Offloading and Multi-GPU Setups. We concentrated on single-GPU scenarios, reflecting common deployment choices where models are intentionally selected to fit available memory for low latency and high performance. Bench360 is ready to be extended with CPU offloading and multi-GPU support, which we plan to explore in subsequent work.

6 CONCLUSION

We introduced Bench360, a novel framework for evaluating local LLM deployments by integrating task-specific quality with system metrics like latency, throughput, and energy consumption. Our work addresses the critical need for a holistic, user-focused benchmark for local inference.

Our results reveal that the optimal configuration is a complex trade-off. We found that while quantization enables larger, more accurate models on memory-constrained hardware, it often incurs significant latency and energy costs. Furthermore, the best-performing inference engine is highly scenario-dependent: LMDeploy excels at single-request responsiveness, SGLang and vLLM lead in batch throughput on different hardware tiers, and vLLM is the most robust and energy-efficient for multi-user serving.

Ultimately, our findings demonstrate there is no single best setup for local inference. The ideal choice demands a data-driven evaluation of the model, quantization, and engine against specific deployment requirements. Bench360 provides the community with a vital tool to navigate this complex decision space and optimize deployments for their unique needs.

REPRODUCIBILITY STATEMENT

All our source code, experimental configurations and raw results are available under the following anonymous URL: https://anonymous.4open.science/r/bench360-9122The repository is documented and enables the complete reproducibility of our results. Bench360 will be open-sourced under a MIT License. Our intention is to enable other researchers to contribute to Bench360 by contributing new tasks and experimental results for additional hardware platforms and LLMs.

ETHICS STATEMENT

We do not recognize any ethical issues and conflicts with the ICLR Code of Ethics. In fact, by making the trade-offs in local LLM inference more transparent, we are supporting the broader availability of local inference. The use of our benchmark may lead to a temporary increase in energy consumption for the evaluation of different LLMs and their deployment configuration. However, we believe this short-term cost is outweighted by the long-term benefits: Bench360 enables researchers and industry practitioners to make informed decisions about the LLMs to use in production environments, enabling them to find efficient solutions that reduce the energy consumption and its correlated CO2 emissions. The energy source for running our experiments has been based entirely on a 100% renewable energy mix. **Use of Large Language Models (LLMs).** We have used LLMs to improve writing of this paper by drafting paragraphs of text, pasting them into cloud-based LLMs (Gemini, ChatGPT) for revision.

REFERENCES

Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for llm inference. *Proceedings of Machine Learning and Systems*, 6:351–366, 2024.

Mauricio Fadel Argerich and Marta Patiño-Martínez. Measuring and improving the energy efficiency of large language models inference. *IEEE Access*, 2024.

Ishan Pandey Arnav Chavan, Deepak Gupta. Edge llm leaderboard. https://huggingface.co/spaces/nyunai/edge-llm-leaderboard, 2024.

Edward Beeching, Clémentine Fourrier, Nathan Habib, Sheon Han, Nathan Lambert, Nazneen Rajani, Omar Sanseviero, Lewis Tunstall, and Thomas Wolf. Open llm leaderboard. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard, 2023.

Krishna Teja Chitty-Venkata, Sparsh Mittal, Murali Emani, Venkatram Vishwanath, and Arun K Somani. A survey of techniques for optimizing transformer inference. *Journal of Systems Architecture*, 144:102990, 2023.

Krishna Teja Chitty-Venkata, Siddhisanket Raskar, Bharat Kale, Farah Ferdaus, Aditya Tanikanti, Ken Raffenetti, Valerie Taylor, Murali Emani, and Venkatram Vishwanath. Llm-inference-bench: Inference benchmarking of large language models on ai accelerators. In *SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1362–1379. IEEE, 2024.

Jae-Won Chung, Jiachen Liu, Jeff J Ma, Ruofan Wu, Oh Jun Kweon, Yuxuan Xia, Zhiyu Wu, and Mosharaf Chowdhury. The ml. energy benchmark: Toward automated inference energy measurement and optimization. *arXiv preprint arXiv:2505.06371*, 2025.

LMDeploy Contributors. Lmdeploy: A toolkit for compressing, deploying, and serving llm. https://github.com/InternLM/lmdeploy, 2023.

Ray Project Contributors. Llmperf: Scalable and reproducible llm inference benchmarking. https://github.com/ray-project/llmperf, 2024. Accessed: 2025-05-04.

Fahim Dalvi, Maram Hasanain, Sabri Boughorbel, Basel Mousi, Samir Abdaljalil, Nizi Nazar, Ahmed Abdelali, Shammur Absar Chowdhury, Hamdy Mubarak, Ahmed Ali, et al. Llmebench: A flexible framework for accelerating llms benchmarking. *arXiv preprint arXiv:2308.04945*, 2023.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Llm.int8(): 8-bit matrix multiplication for transformers at scale. *arXiv preprint arXiv:2208.07339*, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Hugging Face. Text generation inference (v3.0). https://github.com/huggingface/text-generation-inference/releases/tag/v3.0, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023. URL https://arxiv.org/abs/2210.17323.

Jonathan Fürst, Catherine Kosten, Farhard Nooralahzadeh, Yi Zhang, and Kurt Stockinger. Evaluating the data model robustness of text-to-sql systems based on real user queries. *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain*, 2025.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hugging Face. Text Generation Inference v3 Overview: Chunked KV Caching. https://huggingface.co/docs/text-generation-inference/en/conceptual/chunking, 2025. Accessed 21 Jul 2025.

InternLM Team. LMDeploy Documentation: INT4/INT8 KV Cache. https://lmdeploy.readthedocs.io/en/latest/quantization/kv_quant.html, 2024. Accessed 21 Jul 2025.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Kiran Kasichayanula, Dan Terpstra, Piotr Luszczek, Stan Tomov, Shirley Moore, and Gregory D Peterson. Power aware computing on gpus. In *2012 Symposium on Application Accelerators in High Performance Computing*, pp. 64–73. IEEE, 2012.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023a.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. *arXiv preprint arXiv:2309.06180*, 2023b. URL https://arxiv.org/abs/2309.06180. SOSP'23 paper.

Shiyao Li, Xuefei Ning, Luning Wang, Tengxuan Liu, Xiangsheng Shi, Shengen Yan, Guohao Dai, Huazhong Yang, and Yu Wang. Evaluating quantized large language models, 2024. URL https://arxiv.org/abs/2402.18158.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Alexander Cosgrove, Christopher D Manning, Christopher Re, Diana Acosta-Navas, Drew Arad Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue WANG, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri S. Chatterji, Omar Khattab, Peter

Henderson, Qian Huang, Ryan Andrew Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=iO4LZibEqW. Featured Certification, Expert Certification.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han. Awq: Activation-aware weight quantization for llm compression and acceleration. *arXiv*, 2023.

Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, Guenther Schmuelling, Hanlin Tang, et al. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

Iftitahu Nimah, Meng Fang, Vlado Menkovski, and Mykola Pechenizkiy. Nlg evaluation metrics beyond correlation analysis: An empirical metric preference checklist. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

NVIDIA. NVIDIA A10 Tensor Core GPU Datasheet. https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a10/pdf/a10-datasheet.pdf, 2020. Accessed on July 9, 2025. Peak performance figures for INT4 are explicitly given with and without sparsity.

NVIDIA. NVIDIA A30 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/products/a30-gpu/, 2021. Accessed on July 9, 2025. Specifications often listed with sparsity, but widely understood that non-sparse performance is half.

NVIDIA. NVIDIA L4 Tensor Core GPU. https://www.nvidia.com/en-us/data-center/l4/, 2023. Accessed on July 9, 2025. Specifications often listed with sparsity, but widely understood that non-sparse performance is half.

NVIDIA Corporation. Metrics — nvidia nim llms benchmarking. https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html, June 2025. Accessed: 2025-09-23.

Youngsuk Park, Kailash Budhathoki, Liangfu Chen, Jonas M Kübler, Jiaji Huang, Matthäus Kleindessner, Jun Huan, Volkan Cevher, Yida Wang, and George Karypis. Inference optimization of foundation models on ai accelerators. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 6605–6615, 2024.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*, 2018.

Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, Ramesh Chukka, Cody Coleman, Sam Davis, Pan Deng, Greg Diamos, Jared Duke, Dave Fick, J. Scott Gardner, Itay Hubara, Sachin Idgunji, Thomas B. Jablin, Jeff Jiao, Tom St. John, Pankaj Kanwar, David Lee, Jeffery Liao, Anton Lokhmotov, Francisco Massa, Peng Meng, Paulius Micikevicius, Colin Osborne, Gennady Pekhimenko, Arun Tejusve Raghunath Rajan, Dilip Sequeira, Ashish Sirasao, Fei Sun, Hanlin Tang, Michael Thomson, Frank Wei, Ephrem Wu, Lingjie Xu, Koichi Yamada, Bing Yu, George Yuan, Aaron Zhong, Peizhao Zhang, and Yuchen Zhou. Mlperf inference benchmark, 2019.

Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL https://www.aclweb.org/anthology/P17-1099.

Sithursan Sivasubramaniam, Cedric E Osei-Akoto, Yi Zhang, Kurt Stockinger, and Jonathan Fürst. Sm3-text-to-query: Synthetic multi-model medical text-to-query benchmark. *Advances in Neural Information Processing Systems*, 37:88627–88663, 2024.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Arya Tschand, Arun Tejusve Raghunath Rajan, Sachin Idgunji, Anirban Ghosh, Jeremy Holleman, Csaba Kiraly, Pawan Ambalkar, Ritika Borkar, Ramesh Chukka, Trevor Cockrell, et al. Mlperf power: Benchmarking the energy efficiency of machine learning systems from microwatts to megawatts for sustainable ai. *arXiv preprint arXiv:2410.12032*, 2024.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921, 2018.

Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

Li Zhang, Youhe Jiang, Guoliang He, Xin Chen, Han Lv, Qian Yao, Fangcheng Fu, and Kai Chen. Efficient mixed-precision large language model inference with turbomind. *arXiv preprint arXiv:2508.15601*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2024a. URL `https://arxiv.org/abs/2312.07104`. v2, last revised 6 Jun 2024.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Livia Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E Gonzalez, et al. Sglang: Efficient execution of structured language model programs. *Advances in neural information processing systems*, 37: 62557–62583, 2024b.

A System Performance Metrics

A.1 Latency

We define three related latency measurements: TTFT (time-to-first-token) is the delay between issuing a request at time t_{request} and receiving the first token at $t_{\text{first token}}$; ATL (average token latency) is the total generation time T_{gen} divided by the number of tokens N_t; and GL (generation latency) multiplies ATL by N_t, which recovers the full T_{gen}:

$$\text{TTFT} = t_{\text{first token}} - t_{\text{request}} \tag{1}$$

$$\text{TPOT} = \frac{T_{\text{gen}}}{N_t} \tag{2}$$

$$\text{GL} = \text{TPOT} \times N_t = T_{\text{gen}} \tag{3}$$

A.2 Throughput

We quantify generation speed in terms of tokens per second (TPS) and sentences per second (TPS). Here, N_t is the total number of tokens generated, N_s is the total number of sentences generated, and

T_{gen} is the generation time in seconds:

$$\text{TPS} = \frac{N_t}{T_{\text{gen}}} \tag{4}$$

$$\text{SPS} = \frac{N_s}{T_{\text{gen}}} \tag{5}$$

A.3 GPU Memory Usage

We capture both average ($\overline{m}$) and peak ($m_{\max}$) GPU memory consumption over the measurement period. Here, m_i is the memory usage (MB) at sample i, and n is the total number of samples:

$$\overline{m} = \frac{1}{n}\sum_{i=1}^{n} m_i \tag{6}$$

$$m_{\max} = \max_{1\le i\le n} m_i \tag{7}$$

A.4 GPU Utilization

We capture both average ($\overline{u}$) and peak ($u_{\max}$) GPU utilization over the measurement period. Here, u_i is the GPU utilization percentage at sample i, and n is the total number of samples:

$$\overline{u} = \frac{1}{n}\sum_{i=1}^{n} u_i \tag{8}$$

$$u_{\max} = \max_{1\le i\le n} u_i \tag{9}$$

B System Cost & Resource Metrics

B.1 Power & Energy

This set of metrics defines average and peak power and derives energy consumption. Here, p_i is the power reading (W) at sample i, n is the total number of samples, T_{gen} is the generation time in seconds, N_t is the number of tokens generated, and N_s is the number of sentences generated:

$$\overline{p} = \frac{1}{n}\sum_{i=1}^{n} p_i \tag{10}$$

$$p_{\max} = \max_{1\le i\le n} p_i \tag{11}$$

$$E_{\text{Wh}} = \frac{\overline{p} \times T_{\text{gen}}}{3600} \tag{12}$$

$$E_{\text{J}} = E_{\text{Wh}} \times 3600 = \overline{p} \times T_{\text{gen}} \tag{13}$$

$$E_{\text{token}} = \frac{E_{\text{J}}}{N_t} \tag{14}$$

$$E_{\text{sentence}} = \frac{E_{\text{J}}}{N_s} \tag{15}$$

B.2 Model Size & Overhead

Overhead captures the additional GPU memory consumed at runtime beyond the model's static file size. Here, M_{size} denotes the total size of the model's weight files on disk (in MB), and $\overline{m}$ is the average GPU memory footprint (in MB) measured during inference:

$$\text{Overhead} = \max\big(\overline{m} - M_{\text{size}},\, 0\big) \tag{16}$$

C System Deployment & Infrastructure Metrics

C.1 Cold Start Time

T_{cold} measures the total latency of a fresh inference invocation. Here, T_{startup} denotes the time to initialize the inference framework, T_{load} is the time to load the model weights into memory, and TTFT (time-to-first-token) is the latency from request issuance to receiving the first token:

$$T_{\text{cold}} = T_{\text{startup}} + T_{\text{load}} + \text{TTFT} \tag{17}$$

D Hardware Setup

Experiments were performed on two full-virtualized KVM guests under Ubuntu. Below is a summary of their key specifications:

Table 6: Hardware specifications of the three machines used in our experiments.

Component	**Machine 1**	**Machine 2**	**Machine 3**
CPU	AMD EPYC-Milan (1×16 cores)	Intel Broadwell (1×16 cores)	Intel Ice Lake (30 vCPUs)
Memory	47 GiB	31 GiB	200 GiB
Storage	500 GB NVMe	812 GB NVMe	1.4 TB SSD
GPU	NVIDIA L4, 24 GiB (CUDA 12.8)	NVIDIA A30, 24 GiB (CUDA 12.9)	NVIDIA A10, 24 GiB (CUDA 12.9)
OS & Kernel	Ubuntu 6.8.0-58-generic	Ubuntu 5.15.0-139-generic	Ubuntu 6.8.0-58-generic

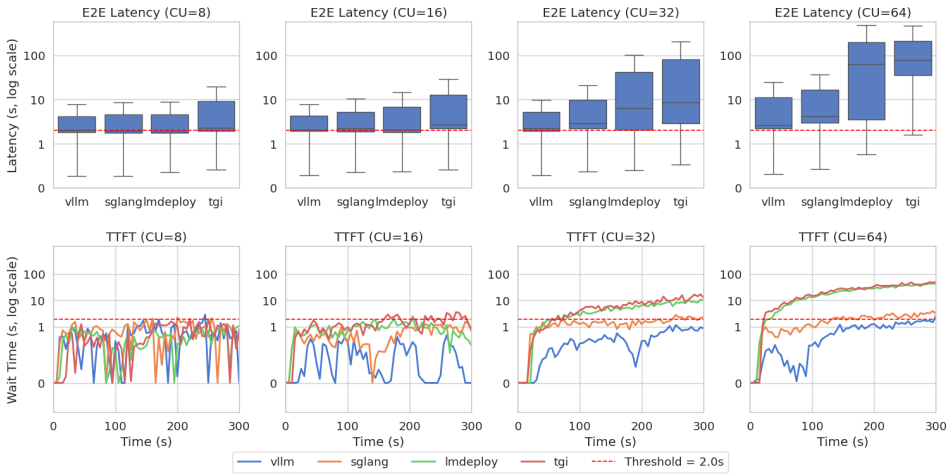
E Additional Experimental Results

Feature	**vLLM**	**SGLang**	**LMDeploy**	**TGI**
Maintainer	UC Berkeley	LMSYS	OpenMMLab	Hugging Face
License	Apache 2.0	Apache 2.0	Apache 2.0	Apache 2.0
Quantization	FP16, GPTQ, AWQ	FP16, INT4	GPTQ, AWQ	FP16, INT8
KV Cache	PagedAttn	Fine-grained	Prefill-optim.	CUDA-based
Streaming	Yes	Yes	Yes	Yes
Multi-Model	No	Yes	Yes	Yes
Batching	Token-level	Static	Token-level	Token-level
API	OpenAI-comp.	OpenAI-comp.	OpenAI-comp.	OpenAI-comp.
Docker	Official image	Official image	Official image	Official image

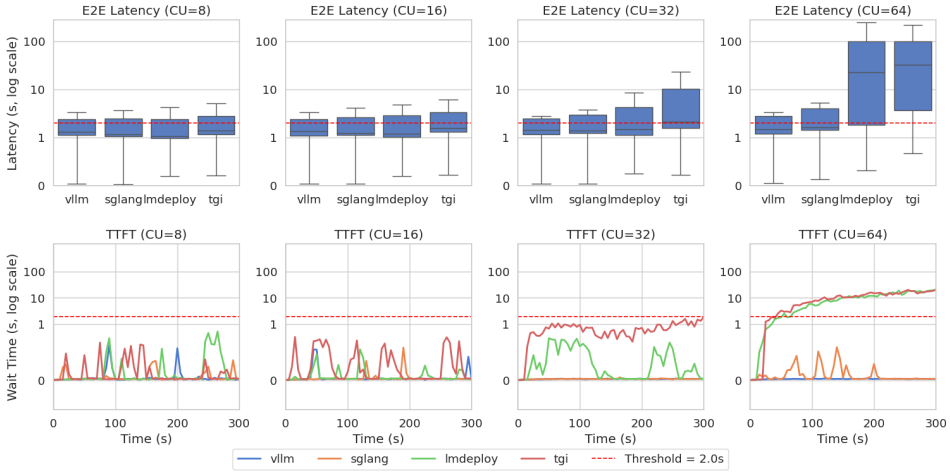
Table 7: Comparison of local LLM inference engines used in our study: vLLM Kwon et al. (2023b), SGLang Zheng et al. (2024a), LMDeploy InternLM Team (2024), and TGI Hugging Face (2025).

Spec	**L4**	**A10**	**A30**
GPU Architecture	Ada Lovelace	Ampere	Ampere
CUDA Cores	7,424	9,216	10,752
Memory	24 GB GDDR6	24 GB GDDR6	24 GB HBM2
Memory Bandwidth	∼300 GB/s	∼600 GB/s	∼933 GB/s
FP16 Tensor TFLOPs	242 TFLOPs*	125 / 250 TFLOPs*	165 / 330 TFLOPs*
INT8 Tensor TOPS	485 TOPS*	250 / 500 TOPS*	330 / 661 TOPS*
TDP	72 W	150 W	165 W
BF16 Support	Yes	Yes	Yes
INT4 GPTQ Support	Yes	Yes	Yes
Form Factor	Low-profile (PCIe)	Full-height (PCIe)	Dual-slot (PCIe)

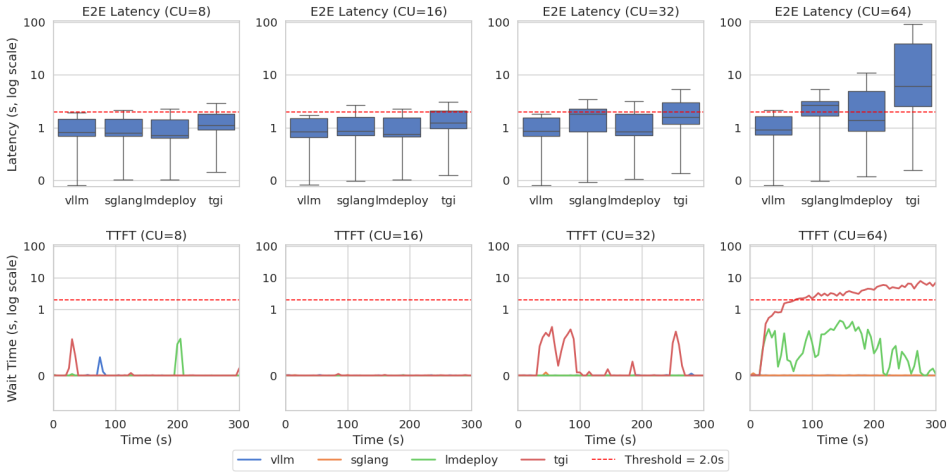
Table 8: Comparison of L4 NVIDIA (2023), A10 NVIDIA (2020), and A30 NVIDIA (2021) GPUs for LLM inference. Values marked with * include structured sparsity (2:4).



(a) L4



(b) A10



(c) A30

Figure 5: **Server-side concurrency across GPUs.** Rows in each panel show E2E (top, log scale) and TTFT (bottom) vs. concurrency. Red dashed lines indicate QoS thresholds: 2 s TTFT and 6 s E2E.