WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

LLM Bench – AI Powered Local LLM Hardware Compatibility and Performance Benchmarking Platform

Ms. Perachiselvi M[1], Kamaleswaran S N[2]

[a]Department of Artificial Intelligence and Machine Learning, Sri Shakthi Institute of Engineering and Technology, Coimbatore – 641 062, Tamil Nadu, India

A B S T R A C T :

Running open-weight Large Language Models (LLMs) on consumer hardware is hampered by trial and error: users download 8–30 GB of weights only to meet Out-Of-Memory (OOM) failures, unbudgeted Key-Value (KV) cache overheads, or unsuitable quantization choices. This paper presents LLM Bench, an open-source platform that verifies hardware compatibility and predicts token generation speed before any model is downloaded. A standalone Windows scanner profiles the GPU, VRAM, system RAM, CPU cores, and storage in about 7 s without administrative privileges. A backend engine converts GGUF file sizes (Q4_K_M, Q5_K_M, Q8_0, FP16) into loaded memory footprints with a 15% safety margin and assigns a Recommended, Minimum, or Not Recommended verdict. A physics-informed constrained linear regression model, PhysicsLinear, predicts tokens per second under full GPU offload, CPU fallback, and partial offloading. Under Leave-One-Group-Out (LOGO) cross-validation it reached a Mean Absolute Percentage Error (MAPE) of 42.01%, the lowest among the candidates evaluated. The platform combines a React 18 front end, a FastAPI back end, PostgreSQL on Supabase, and ephemeral 15-minute sessions that avoid persisting identifying user information.

Keywords: LLM inference, GGUF quantization, hardware compatibility, KV-cache, token generation speed prediction, constrained linear regression

Introduction

Open-weight Large Language Models (LLMs) such as Llama (Touvron et al., 2023), Mistral, Qwen, and DeepSeek can now run on personal computers, developer workstations, and edge devices through runtimes such as llama.cpp (Gerganov, 2023) and Ollama (Ollama Development Team, 2024). Local execution offers data privacy, offline operation, zero per-token API cost, and freedom from vendor lock-in. It also exposes a hardware problem. Autoregressive decoding is memory-bandwidth-bound, so a model whose weights and Key-Value (KV) cache exceed GPU memory either fails with an Out-Of-Memory (OOM) error or spills into system RAM over the PCI Express (PCIe) bus, where throughput can collapse from dozens of tokens per second to below one.

Quantization, notably the GGUF format with variants such as Q4_K_M, Q5_K_M, and Q8_0 (Dettmers et al., 2023), shrinks 16-bit weights to 4–8 bits, but it leaves the user with a selection problem. Static file sizes do not reflect runtime memory, because the KV-cache grows with context length and the host operating system reserves roughly 0.5–2.5 GB of VRAM. Existing diagnostic tools report raw specifications, and cloud profilers may require invasive access. Users therefore download tens of gigabytes before learning whether a model fits.

This paper presents LLM Bench, an end-to-end platform that answers the question before the download. Its contributions are: (i) a non-invasive desktop scanner that profiles GPU, VRAM, RAM, CPU, and storage in about 7 s without administrative privileges; (ii) a dynamic quantization fit model that adds a calibrated 15% safety margin and an operating-system VRAM reserve; (iii) PhysicsLinear, a physics-informed, non-negative linear regression model for tokens-per-second prediction; and (iv) a privacy-first web service with ephemeral 15-minute sessions, an interactive model explorer, and a multi-metric leaderboard.

Literature review

1.1. Local deployment

Frameworks such as Ollama, llama.cpp, vLLM, and LM Studio make it possible to run multi-billion-parameter models locally, giving data sovereignty, deterministic latency, and no per-token billing. During decoding every generated token streams all parameters through the memory bus, so viability is governed by VRAM capacity and bandwidth rather than raw TFLOPS. These tools run models once downloaded but give no pre-download guidance on whether the hardware can sustain them.

1.2. Quantization

Post-training quantization lowers precision from FP16 to 8-, 5-, or 4-bit integers. The GGML/GGUF specification developed by Gerganov (2023) is the de facto standard for local use, and k-quants such as Q4_K_M and Q5_K_M apply mixed precision to retain perplexity and MMLU (Hendrycks et al., 2021) capability while shrinking weights by up to 70%. Choosing correctly still requires accounting for both the parameter footprint and KV-cache growth: Q8_0 may fail on a 6 GB GPU where Q4_K_M of the same model runs at full memory bandwidth.

1.3. Memory hierarchy

Consumer GPUs pair GDDR6 VRAM (about 250–500 GB/s) with system RAM (DDR4/DDR5, 25–60 GB/s) reached over PCIe (16–32 GB/s). When a model exceeds free VRAM, runtimes offload the remaining layers to system RAM and latency becomes dominated by those layers, a non-linear bandwidth cliff that an accurate predictor must model.

1.4. Latency estimation

Analytical roofline models (Williams, Waterman, & Patterson, 2009) give throughput upper bounds but underestimate real latency because they ignore kernel-launch overhead, memory fragmentation, and host contention. Generic non-linear regressors such as multilayer perceptrons or random forests can predict physically impossible values, for example negative latencies, outside their training distribution. Combining first-principles bandwidth-ratio features with a constrained linear regression keeps predictions monotonic, interpretable, and bounded.

1.5. Research gap

Table 1 summarizes the approaches in use. None connects non-invasive hardware inspection, dynamic quantization sizing, and empirical speed prediction in one zero-install system that returns transparent verdicts without tracking private user data.

Table 1 - Summary of existing methods for LLM hardware compatibility and inference.

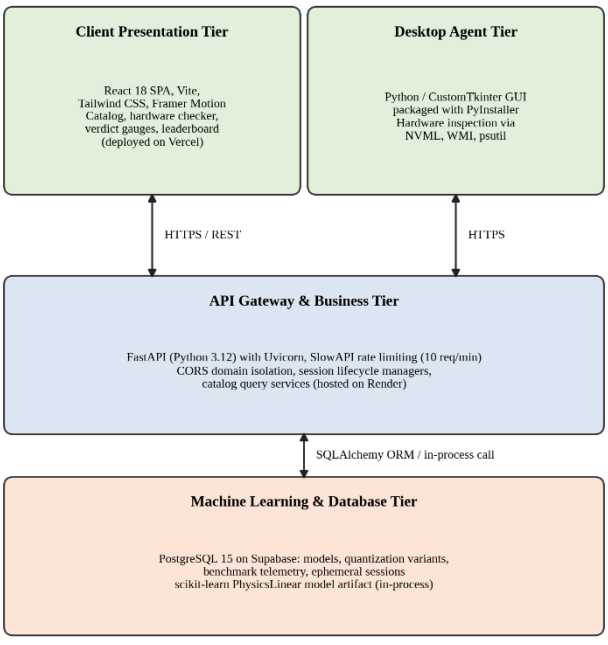
Approach	Limitations
Manual trial and error	Wastes bandwidth; frequent OOM crashes; no pre-flight guidance
Hardware utilities (CPU-Z, DXDiag)	Raw telemetry; no model-weight or KV-cache awareness
Static parameter calculators	Ignore KV-cache, OS VRAM reserve, and PCIe offload bottlenecks
Synthetic GPU benchmarks (3DMark, FurMark)	Do not reflect memory-bound token decoding
Pure ML latency predictors	Data leakage; unphysical negative latencies outside the training range

Proposed system and methodology

1.6. System overview

LLM Bench is a full-stack web and desktop platform organized in four tiers (Fig. 1). The client presentation tier is a React 18 single-page application (Meta Open Source, n.d.). The desktop agent tier is a Python/CustomTkinter scanner packaged with PyInstaller. The API tier is a FastAPI service (Ramírez, n.d.) on Python 3.12 with Uvicorn, SlowAPI rate limiting, and CORS isolation. The data tier is PostgreSQL 15 on Supabase (Supabase, n.d.), while the scikit-learn (Pedregosa et al., 2011) PhysicsLinear model runs inside the backend process for sub-millisecond prediction.

Fig. 1 - System architecture of LLM Bench.



1.7. Desktop scanner and ephemeral sessions

The scanner is a single-file Windows executable. It detects GPUs through the NVIDIA Management Library (NVML) and Windows Management Instrumentation (WMI), reads RAM and CPU cores with psutil, and checks free storage, all without administrative privileges. It first requests a session token from POST /api/scan-sessions, then submits an anonymous hardware payload to POST /api/hardware-scan over HTTPS. Because the cloud backend may be cold-starting, the scanner shows a notification after 2.5 s and allows a 70 s timeout with up to three retries. Sessions expire after 15 minutes, records older than one hour are purged, public write endpoints are limited to 10 requests per minute per IP address, and production CORS is restricted to the application's Vercel domains. Reported GPUs are resolved against a lookup table of 53 consumer models.

1.8. Dynamic quantization fit model

The loaded memory footprint M_{load} (GB) is obtained from the download file size S_{file} by an empirical linear map with a 15% safety margin against fragmentation, and the VRAM available to the model is the physical VRAM V_{gpu} less a fitted 1.85 GB operating-system display reserve:

$$M_{load} = (1.100 \cdot S_{file} + 0.100) \times 1.15 \qquad (1)$$

$$V_{avail} = max(0,\ V_{gpu} - 1.85) \qquad (2)$$

The verdict follows from these quantities. *Recommended* means $M_{\text{load}} \le V_{\text{avail}}$ (full GPU offload). *Minimum* means the model fits when partially offloaded to system RAM, that is $M_{\text{load}} \le V_{\text{avail}} + 16.0$ GB. Otherwise the verdict is *Not Recommended*. The context-length KV-cache is added to the footprint according to layer count and attention dimensions.

1.9. PhysicsLinear speed prediction

Let $f = \min(1,\ V_{\text{avail}} / M_{\text{load}})$ be the fraction of weights resident in VRAM. Two latency features capture the bandwidth asymmetry between GPU memory (B_{gpu}) and the host path (B_{cpu}):

$$x_{gpu} = f \cdot M_{load} / B_{gpu} \qquad (3)$$

$$x_{cpu} = (1 - f) \cdot M_{load} / B_{cpu} \qquad (4)$$

A non-negative linear regression without intercept then predicts per-token latency:

$$t_{tok} = c_1 \cdot x_{gpu} + c_2 \cdot x_{cpu} \qquad (5)$$

The fitted coefficients are $c_1 = 1000.0$ and $c_2 = 1666.7$. The prediction is floored at 0.1 ms and converted to tokens per second, so it can never be negative or unbounded. Training used empirical benchmark telemetry collected with Ollama across full GPU offload, CPU fallback, and partial offloading, and models were compared with Leave-One-Group-Out (LOGO) cross-validation grouped by unique hardware–model configuration to prevent leakage between folds.

1.10. Processing flow

Fig. 2 shows the end-to-end flow: the scanner (or web wizard) creates a session, the payload is stored and matched against the GPU table, the footprint and headroom are computed, PhysicsLinear predicts speed, and the verdict is delivered to the dashboard.

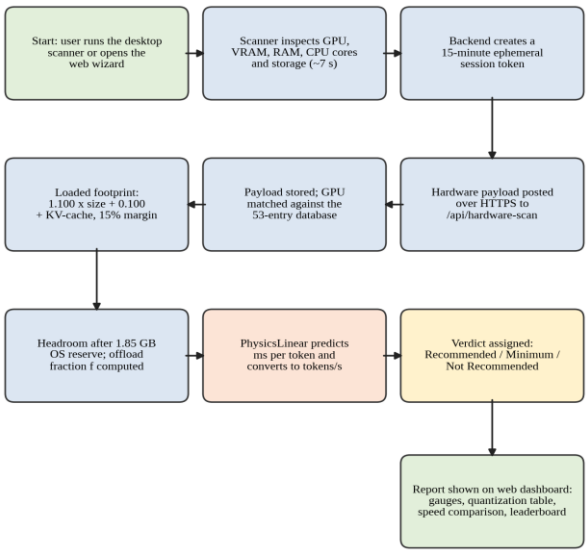


Fig. 2 - Process flow of hardware scan and compatibility verdict.

Implementation and results

1.11. Implementation

The front end uses React 18, Vite, Tailwind CSS, and Framer Motion and is deployed on Vercel. The back end uses FastAPI with SQLAlchemy 2.0 and Pydantic v2 and is hosted on Render. The PostgreSQL schema has five tables (hardware_sessions, models, model_quants, benchmark_results, and gpus) in Third Normal Form. The catalog holds more than 20 models and 47 quantization variants, filterable by size class, capability, and license, with a leaderboard ranking models by measured speed, MMLU quality, and memory efficiency, and reporting sample size and confidence interval for each quality score. Quality assurance comprises 44 backend pytest tests, 11 scanner tests, and a GitHub Actions smoke-test gate that polls the production /api/health endpoint up to 12 times at 15 s intervals after each merge to the main branch.

1.12. Scanner, verdict, and security results

The scanner profiled GPU, VRAM, RAM, CPU cores, and storage in about 7 s, under the 10 s target. In the verdict engine, a 4 GB GPU was correctly advised to use Q4_K_M for small-parameter models while 7B/8B models were flagged Not Recommended. Calling POST /api/scan-sessions or POST /api/hardware-scan more than 10 times per minute from one IP address returned HTTP 429, and cross-origin requests were confirmed to be restricted to the application's Vercel domains.

1.13. Speed prediction accuracy

Table 2 reports LOGO cross-validation results for two baselines and two learned candidates.

Table 2 - LOGO cross-validation metrics of candidate speed prediction models.

Model	MAE (tokens/s)	MAPE (%)	R^2 score	Status
MeanBaseline	34.50	52.96	−0.8143	Baseline
TheoryBaseline	46.24	50.64	−4.2591	Baseline
PhysicsLinear	**33.62**	**42.01**	**−1.5036**	**Selected**
RandomForest	32.30	48.33	−0.3345	Candidate

PhysicsLinear achieved the lowest MAPE (42.01%), a 10.95 percentage-point improvement over the mean baseline, while guaranteeing non-negative latency coefficients and monotonic behavior. The results also show the limits of the approach. RandomForest has a slightly lower MAE (32.30 versus 33.62 tokens/s), and every model has a negative R^2, including PhysicsLinear (−1.5036), which is lower than that of MeanBaseline (−0.8143). The error is therefore still substantial, and the model is best read as a physically consistent estimate for ranking and screening rather than a precise throughput forecast. We selected it on MAPE and interpretability, since a percentage error treats slow offloaded configurations and fast GPU-resident ones on a comparable scale.

Conclusion

LLM Bench replaces the download-and-hope workflow for local LLMs with a zero-install scan that returns a memory-aware verdict and a speed estimate in about 7 s. Its quantization fit model accounts for KV-cache growth, a 15% safety margin, and the operating-system VRAM reserve, and its PhysicsLinear predictor reaches a 42.01% LOGO MAPE with physically bounded output. Rate limiting, CORS controls, cold-start handling, and automated smoke tests support reliable operation, and ephemeral 15-minute sessions preserve user privacy. Future work includes extending the benchmark dataset to AMD ROCm and Apple Silicon unified memory, supporting multi-GPU tensor parallelism, adding one-click Ollama model pull and launch from the browser, and automated context-length testing.

REFERENCES :

- Dettmers, T., et al. (2023). QLoRA: Efficient finetuning of quantized LLMs. *Advances in Neural Information Processing Systems (NeurIPS), 36.*
- Gerganov, G. (2023). llama.cpp: Port of Facebook's LLaMA model in C/C++. *GitHub Repository.* https://github.com/ggerganov/llama.cpp
- Hendrycks, D., et al. (2021). Measuring massive multitask language understanding (MMLU). *International Conference on Learning Representations (ICLR).*
- Meta Open Source. (n.d.). *React documentation.* https://react.dev
- Ollama Development Team. (2024). *Ollama: Get up and running with large language models locally.* https://ollama.com
- Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research, 12,* 2825-2830.
- Ramírez, S. (n.d.). *FastAPI documentation.* https://fastapi.tiangolo.com
- Supabase. (n.d.). *Supabase documentation (PostgreSQL and Storage).* https://supabase.com/docs
- Touvron, H., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288.*
- Williams, S., Waterman, A., & Patterson, D. (2009). Roofline: An insightful visual performance model for multicore architectures. *Communications of the ACM, 52(4),* 65-76.