LLMVisor: A Real-Time Latency Attribution Model for Multi-Tenant LLM Serving

Shuowei Jin[1]* **Xueshen Liu**[1]* **Jiaxin Shan**[2] **Le Xu**[2]
Tieying Zhang[2] **Liguang Xie**[2] **Z. Morley Mao**[1]
[1]**University of Michigan** [2]**ByteDance**

Abstract

As LLM inference shifts to multi-tenant GPU clusters, co-batching improves throughput but obscures per-tenant usage and limits control. Enabling fractional sharing of the inference engine requires a real-time, per-request attribution primitive that is accurate and light enough to run inside the scheduling loop. We present *LLMVisor*, a roofline-guided *latency attribution* model that captures the memory-bound and compute-bound phases via a concise piecewise-linear form over features proportional to FLOPs and memory I/O traffic. *LLMVisor* decomposes batch latency into additive, per-request shares and runs efficiently at μs-scale. We evaluate *LLMVisor* across Llama3.1-8B and Qwen2.5-14B/32B on A100/H100 under varying tensor parallelism and workload mixes. Compared to a token-count baseline, *LLMVisor* attains near-perfect R^2 and reduces relative error by up to $2.5\times/3.3\times$ (p90/p99) for prefill and $3.5\times/4.4\times$ for decode, despite batching variability and sequence divergence.

1 Introduction

Large language models (LLMs) now power search, coding assistants, and conversational systems at scale [4, 7, 5, 8, 20, 21, 23]. To keep accelerators busy, providers co-batch heterogeneous requests via shared public endpoints [16, 2, 19]. While batching boosts throughput, it leaves tenants with opaque, provider-tuned scheduling and little leverage to allocate or account for their own usage. The common alternative—renting dedicated GPU servers—squanders parallel hardware on interactive, small-batch workloads; for example, GPT-J 6B on A100 achieves only 0.4% SM utilization at batch size 1 and remains low even for 1,024-token sequences [10].

We argue for virtualizing the *inference engine*: tenants reserve fractional shares of GPU time and submit requests into shared batches, akin to CPU isolation via VMs, cgroups, and containers [3, 12, 13], and to credit/burst offerings (e.g., AWS `t3.*`) [1]. Hardware partitioning (e.g., NVIDIA MIG) provides strong isolation but is inflexible for LLM memory footprints [15]. Realizing a software path to fractional sharing hinges on a missing primitive: a *real-time, per-request latency attribution model* accurate enough to guide scheduling and light enough to run in the critical path without harming batching efficiency. Such attribution directly enables SLO-aware admission (curbing tails by rejecting or deferring requests whose shares would exceed budgets), transparent post-hoc accounting, and fast "what-if" planning (e.g., alternative batch sizes or mixes) while preserving the benefits of co-batching.

Designing this primitive is difficult for several reasons. First, co-batching couples requests: their costs are not trivially separable. Second, LLM inference has phase-specific bottlenecks, compute-bound *prefill* versus memory-bound *decode*, that require different treatments. Third, costs scale nonlinearly with sequence length (e.g., quadratic self-attention) and depend on context length (KV-cache loads) and batch size. Fourth, throughput shifts with model architecture, tensor-parallel layout, and hardware. Finally, the scheduler operates at millisecond granularity, so attribution must run at μs scale to be

*Equal contribution.

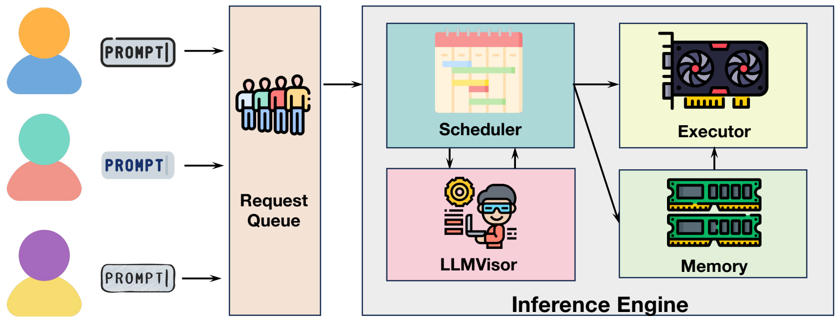

Figure 1: A typical workflow in modern large language model (LLM) inference engines can be summarized as follows. Upon arrival, each request is enqueued in a request buffer. At each inference step, the scheduler selects requests from the queue for execution. *LLMVisor* operates during the scheduling stage by providing GPU time attribution, enabling the scheduler to enforce multi-tenant fairness according to reserved GPU time quotas. Once scheduled, the inference engine dispatches the requests to the GPU, where the corresponding KV cache and model weights are loaded to perform the inference computation.

viable. Black-box ML predictors can fit latency trends but lack additivity and are too slow for the loop [22]; naive token-count proxies [17] ignore context and batching effects, yielding poor high-percentile accuracy.

We present *LLMVisor*, a roofline-guided *latency attribution* model for multi-tenant LLM serving. *LLMVisor* captures prefill and decode with a concise piecewise-linear form over features proportional to FLOPs and memory I/O traffic—explicitly encoding quadratic self-attention, context-length–driven KV-cache traffic, and batch-size effects—so end-to-end batch latency decomposes into additive, per-request shares. The resulting structure supports online μs-scale "what-if" queries for admission control and budgeting while preserving batching efficiency. A lightweight prototype integrated into vLLM runs in the scheduling path with negligible overhead and achieves low error across diverse models and GPUs.

Our contributions are threefold:

• **Inference-aware modeling:** An interpretable, roofline-guided, piecewise-linear formulation that embeds LLM execution mechanics: compute-bound prefill vs. memory-bound decode, quadratic self-attention, KV-cache traffic via context length, and batch-size utilization effects. The model is additive by design, yielding closed-form per-request latency shares suitable for multi-tenant control.

• **Efficiency:** Microsecond-level runtime per scheduling step (100x faster than common ML predictors), enabling deployment *inside* the scheduler without disrupting batching.

• **Accuracy and generality:** Across Llama3.1-8B and Qwen2.5-14B/32B on A100/H100 with varying tensor parallelism and workload mixes, *LLMVisor* attains near-perfect R^2 and reduces relative error by up to $2.5\times/3.3\times$ (p90/p99) for prefill and $3.5\times/4.4\times$ for decode.

2 *LLMVisor* Design

Design requirements. The state-of-the-art inference engine workflow is illustrated in Fig. 1. *LLMVisor* assists the scheduler in ensuring that each tenant receives its reserved share of GPU time. To be practical in high-frequency scheduling loops (e.g., every decode step), *LLMVisor* must combine three properties: it must be ***accurate***, providing reliable predictions of end-to-end batch latency; it must support ***attribution fidelity***, decomposing that latency into per-request and per-tenant contributions; and it must be ***efficient***, introducing negligible overhead so that scheduling can operate at each step[2]. *LLMVisor* is designed to satisfy all three simultaneously.

Problem formulation. Formally, let $\mathcal{B} = \{1, \ldots, |\mathcal{B}|\}$ denote a batch of requests and $u(i)$ the tenant of request i. We seek a model f that (i) predicts the end-to-end latency of the batch, $f(\mathcal{B}) \approx T_{\mathcal{B}}$,

[2]Step is the minimal execution unit in LLM inference. During decoding, one step corresponds to all active requests in the batch generating a single token. During prefill, one step corresponds to all requests in the batch completing their entire prompt processing.

and (ii) assigns each request a nonnegative share $f(i \mid \mathcal{B}) \approx t_i$ such that $\sum_{i \in \mathcal{B}} f(i \mid \mathcal{B}) = f(\mathcal{B})$. These per-request attributions capture each request's latency contribution and aggregate naturally to per-tenant usage $L_u = \sum_{i\colon u(i)=u} f(i \mid \mathcal{B})$, supporting fairness, admission control, and billing.

Related work. Prior studies on LLM latency prediction follow two main approaches. The first uses ML predictors [6], which can be accurate but cannot attribute latency to individual requests, limiting their usefulness for multi-tenant scheduling. The second relies on analytical, roofline-style modeling [22], classifying inference as compute- or memory-bound and estimating latency from FLOPs and memory traffic relative to GPU peaks. While interpretable, these models are often inaccurate in practice due to modeling assumptions and time-varying GPU throughput, and they do not provide lightweight, per-request attribution in real time. VTC [17] attributes usage by token counts, which is imprecise because compute and I/O vary with token position [9].

2.1 Piecewise roofline-guided linear latency model

LLMVisor builds on the intuition of roofline analysis but adapts it for higher accuracy and real-time use. Rather than hand-computing FLOPs and memory bytes from model architecture and dividing by peak GPU rates, we construct a small set of request-level features proportional to FLOPs and I/O traffic and fit their coefficients using short warm-up profiling runs. This preserves interpretability while improving accuracy and keeping runtime cost negligible.

Let p_i denote the number of tokens *to process* for request i in the current step (all prompt tokens during prefill; $p_i = 1$ during decode), let c_i denote the *context* tokens that must be attended and whose KV cache must be loaded (zero in prefill; prior prompt length in decode), and let $\mathcal{B}$ denote the batch size. As $\sum_i p_i$ increases, the feed-forward network (FFN) transitions from memory-bound to compute-bound. Thus, we model both prefill and decode latency using a two-segment linear form with different coefficient sets $(\beta_\star, \{a_1, a_2, a_3, a_4\}_\star)$.

$$T_{\mathcal{B}} = \beta + a_1 \sum_i p_i + a_2 \sum_i c_i + a_3 \sum_i p_i^2 + a_4 \sum_i |\mathcal{B}| \tag{1}$$

Here, β captures fixed per-batch costs (e.g., model-weight I/O); $\sum_i p_i$ reflects MLP and causal-attention compute; $\sum_i c_i$ accounts for KV-cache memory traffic; $\sum_i p_i^2$ models quadratic self-attention costs; and $\sum_i |\mathcal{B}|$ captures utilization effects. We fit the coefficients via ordinary least squares on profiling data, yielding four sets of parameters $(\beta_\star, \{a_1, a_2, a_3, a_4\}_\star)$ in total—two for prefill and two for decode.

Because $T_{\mathcal{B}}$ is linear in sums of per-request terms, *LLMVisor* naturally admits **closed-form**, additive per-request attributions via Eq. 1. yielding each request's estimated latency contribution in the critical path.

Additionally, in terms of **efficiency**, Eq. 1 is over 100x faster than ML predictors such as Random Forest. In our measurements, *LLMVisor* operates at the microsecond level per request, whereas more complex ML models like Random Forest require milliseconds for inference. This efficiency makes *LLMVisor* well-suited for integration into LLM inference engine scheduling, which runs at millisecond granularity.

3 Evaluation

In this section, we evaluate *LLMVisor* across a diverse range of real-world LLM serving configurations and compare its predictions against ground truth measurements.

3.1 Experimental Setup

We evaluate *LLMVisor* inside vLLM (v0.7.3) [11] using its internal profiler for per-step ground truth, spanning two server classes (H100 SXM and A100 SXM) with multiple tensor parallel sizes, three open-source models (Llama3.1-8B [14], Qwen2.5-14B/32B [18]), and workloads with 128–2048 concurrent requests and heterogeneous sequences that push total tokens to ninety percent of the engine KV-cache capacity ($\mathcal{C}$). We cover both prefill and decode, report coefficient of determination (r^2) and relative error at the 90th and 99th percentiles, and compare against VTC [17], a token based

Model	Llama3.1-8B			Qwen2.5-14B			Qwen2.5-32B		
Parallelism	A1	H1	H4	H1	H2	H4	A2	H2	H4
VTC R^2	0.998	0.998	0.995	1.000	0.997	0.998	1.000	1.000	1.000
VTC Err_{p90}	0.08	0.09	0.04	0.03	0.14	0.08	0.02	0.03	0.02
VTC Err_{p99}	0.50	0.79	0.16	0.09	1.24	0.57	0.10	0.07	0.11
LLMVisor R^2	1.000	0.999	1.000	1.000	1.000	1.000	1.000	1.000	1.000
LLMVisor Err_{p90}	0.03	0.10	0.01	0.02	0.02	0.01	0.01	0.02	0.03
LLMVisor Err_{p99}	0.14	0.92	0.08	0.08	0.08	0.13	0.05	0.07	0.08

Table 1: Prefill step latency modeling accuracy of VTC and *LLMVisor*.

Model	Llama3.1-8B			Qwen2.5-14B			Qwen2.5-32B		
Parallelism	A1	H1	H2	H1	H2	H4	A2	H2	H4
VTC R^2	0.947	0.958	0.778	0.991	0.948	0.938	0.972	0.991	0.992
VTC Err_{p90}	0.23	0.24	0.42	0.12	0.24	0.27	0.15	0.12	0.11
VTC Err_{p99}	0.50	0.54	0.88	0.18	0.45	0.63	0.29	0.20	0.31
LLMVisor R^2	0.974	0.997	0.995	0.998	0.997	0.996	0.979	0.997	0.999
LLMVisor Err_{p90}	0.10	0.04	0.05	0.04	0.04	0.07	0.11	0.06	0.03
LLMVisor Err_{p99}	0.16	0.08	0.09	0.06	0.07	0.10	0.16	0.10	0.05

Table 2: Decode step latency modeling accuracy of VTC and *LLMVisor*.

proxy that does not model batch size or context length. Further details of the evaluation setup are provided in Sec. A.

3.2 Prefill Latency Modeling

The performance of *LLMVisor* and the VTC baseline on the prefill latency modeling task is presented in Table 1. Across all evaluated models and hardware configurations, *LLMVisor* consistently outperforms VTC in modeling prefill latency.

While VTC achieves reasonably high R^2 values (> 0.995 in most cases), it exhibits much larger relative errors, particularly at p99. Averaged across all configurations (excluding the two blue-marked outliers), VTC's relative error is 0.05 at p90 and 0.30 at p99, while *LLMVisor* reduces these to 0.02 and 0.09, respectively. This corresponds to an average improvement of roughly $2.5\times$ at p90 and over $3.3\times$ at p99. The high R^2 values for both methods indicate that VTC's linear token-based model can capture coarse latency trends in prefill phase. However, by taking into account the square of input tokens, which reflects the self-attention computational complexity, *LLMVisor* is far more reliable for predicting prefill latency in multi-tenant serving environments.

3.3 Decode Latency Modeling

The performance of *LLMVisor* and the VTC baseline on the decode latency modeling task is presented in Table 2. Compared to the prefill phase, decode latency is substantially more challenging to predict. This difficulty is evident in VTC's results: while it achieves moderately strong correlation in most cases ($R^2 > 0.9$), it completely breaks down in others. For example, Llama3.1-8B on H2 yields only $R^2 = 0.778$, indicating that VTC is not even able to capture the overall latency trend. At the same time, its relative errors remain high across the board, averaging 0.21 at p90 and 0.44 at p99, and reaching as high as 0.88 at p99. These results show that a simple linear token-based model is fundamentally inadequate for decode latency, as it ignores the effects of context length and batch size.

In contrast, *LLMVisor* consistently maintains near-perfect fits, with $R^2 > 0.97$ in all cases and typically above 0.995. More importantly, *LLMVisor* reduces relative error dramatically: average p90 error drops from 0.21 (VTC) to 0.06, and average p99 error from 0.44 to 0.10, corresponding to $3.5\times$ and $4.4\times$ improvements, respectively. Even in the more challenging decode phase, where latency is influenced by batching variability and sequence divergence, *LLMVisor* reliably captures both the overall trend and high-percentile behavior.

Overall, *LLMVisor* significantly outperforms linear token-based modeling in both prefill and decode phases, with up to 3.5x and 4.4x improvement on p90 and p99 relative error, respectively. It consistently predicts latency accurately and reliably, which will benefit the scheduling in multi-tenant serving environments.

References

[1] Amazon Web Services. Amazon EC2 Instance Types – Burstable Performance Instances (T3). Online, 2023.

[2] Anthropic. Anthropic API. `https://www.anthropic.com/api`, 2025. [Online].

[3] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5):164–177, 2003.

[4] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[5] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

[6] Saki Imai, Rina Nakazawa, Marcelo Amaral, Sunyanan Choochotkaew, and Tatsuhiro Chiba. Predicting llm inference latency: A roofline-driven ml method. In *Annual Conference on Neural Information Processing Systems*, 2024.

[7] Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[8] Shuowei Jin, Xueshen Liu, Yongji Wu, Haizhong Zheng, Qingzhao Zhang, Atul Prakash, Matthew Lentz, Danyang Zhuo, Feng Qian, and Zhuoqing Mao. Plato: Plan to efficient decode for large language model inference. In *Second Conference on Language Modeling*, 2025.

[9] Shuowei Jin, Xueshen Liu, Qingzhao Zhang, and Z Morley Mao. Compute or load kv cache? why not both? *arXiv preprint arXiv:2410.03065*, 2024.

[10] Yunho Jin, Chun-Feng Wu, David Brooks, and Gu-Yeon Wei. S^3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

[11] 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 29th ACM Symposium on Operating Systems Principles (SOSP)*, 2023. arXiv:2309.06180.

[12] Paul Menage, Paul Jackson, and Christoph Lameter. Cgroups. *Available on-line at: http://www. mjmwired. net/kernel/Documentation/cgroups. txt*, 2008.

[13] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[14] Meta. Introducing llama 3.1: The next generation of open models. `https://ai.meta.com/blog/meta-llama-3-1/`, July 2024. Accessed: 2025-03-29.

[15] NVIDIA Corporation. NVIDIA Multi-Instance GPU (MIG) User Guide. `https://docs.nvidia.com/datacenter/tesla/mig-user-guide/`, 2025. [Online].

[16] OpenAI. OpenAI API. `https://openai.com/api/`, 2025. [Online].

[17] Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 965–988, 2024.

[18] Qwen Team. Qwen2.5: A party of foundation models, September 2024.

[19] The AIBrix Team, Jiaxin Shan, Varun Gupta, Le Xu, Haiyang Shi, Jingyuan Zhang, Ning Wang, Linhui Xu, Rong Kang, Tongping Liu, et al. Aibrix: Towards scalable, cost-effective large language model inference infrastructure. *arXiv preprint arXiv:2504.03648*, 2025.

[20] Yongji Wu, Xueshen Liu, Shuowei Jin, Ceyu Xu, Feng Qian, Z Morley Mao, Matthew Lentz, Danyang Zhuo, and Ion Stoica. Hetermoe: Efficient training of mixture-of-experts models on heterogeneous gpus. *arXiv preprint arXiv:2504.03871*, 2025.

[21] Yongji Wu, Xueshen Liu, Haizhong Zheng, Juncheng Gu, Beidi Chen, Z Morley Mao, Arvind Krishnamurthy, and Ion Stoica. Rlboost: Harvesting preemptible resources for cost-efficient reinforcement learning on llms. *arXiv preprint arXiv:2510.19225*, 2025.

[22] 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.

[23] Zesen Zhao, Shuowei Jin, and Z Morley Mao. Eagle: Efficient training-free router for multi-llm inference. *arXiv preprint arXiv:2409.15518*, 2024.

A Evaluation Setup Details

Our evaluation framework is built on vLLM (v0.7.3) [11], using its internal profiler to capture ground truth latency for each processing step. To ensure a comprehensive assessment, we systematically vary the hardware, models, and workloads.

Hardware We conduct experiments on two distinct server types with five different tensor parallelism settings to cover various deployment scenarios: a. A server with 4 NVIDIA H100 SXM 80GB GPUs, a 104-core vCPU, and 1800GiB of memory. Tensor parallel size 1, 2, and 4, denoted as **H1**, **H2**, **H4**. b. A server with 8 NVIDIA A100 SXM 80GB GPUs, a 240-core vCPU, and 1800GiB of memory. Tensor parallel size 1 and 2, denoted as **A1**, **A2**.

Models We test *LLMVisor* on three popular open-source LLMs of varying sizes to evaluate its generalizability: Llama3.1-8B [14], Qwen2.5-14B [18], and Qwen2.5-32B [18].

Workloads To simulate realistic, multi-tenant serving environments, we generate workloads with varying numbers of concurrent requests, from 128 to 2048. To test performance across different context lengths, we increase the total number of tokens from 4096 to 90% of the engine's maximum KV cache capacity ($\mathcal{C}$). These tokens are then randomly distributed among requests with high variance to ensure a heterogeneous mix of request lengths. Our evaluation covers both the prefill and decode phases of inference.

Metrics We assess the accuracy of resource modeling methods via two metrics: a. Coefficient of Determination (r^2): Quantifies the linear correlation between the predicted and ground truth latencies. A higher r^2 indicates a better fit. b. Relative Error: Measures the percentage difference between predicted and actual latency, reported at the 90th (p90) and 99th (p99) percentiles to evaluate performance.

Baseline We benchmark *LLMVisor* against VTC [17], a baseline model that estimates request latency based on a linear model of token number. Unlike *LLMVisor*, VTC does not account for critical system-level factors such as batch size and context length.