InferenceProfiler: Enhancing the Observability of Performance, Cost, and Energy of LLM Inferencing in the Cloud

Austin Bomhold
abomhold@uw.edu
University of Washington
Tacoma, WA, USA

Xinghan Chen
kirito20@uw.edu
University of Washington
Tacoma, WA, USA

Morteza Nabavinejad
mnabavi@uw.edu
University of Washington
Tacoma, WA, USA

Wes Lloyd
wlloyd@uw.edu
University of Washington
Tacoma, WA, USA

Abstract

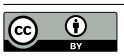
The widespread deployment of Large Language Models (LLMs) requires substantial computational resources, often resulting in high operational costs and energy consumption. Optimizing these complex workloads necessitates precise visibility into how hardware accelerators and host system resources are utilized. This paper presents InferenceProfiler, a software tool that measures and records the resource usage of LLM inferencing tasks. InferenceProfiler is designed to profile the CPU/GPU, memory, disk, network, and power utilization of various tasks, collecting system metrics from the virtual machine (host), cgroup, process, GPU, and inference engine. InferenceProfiler leverages the NVIDIA Management Library (NVML) to capture real-time metrics across varying NVIDIA GPU architectures. InferenceProfiler integrates with vLLM, an open-source high-throughput inference engine for serving LLMs, to correlate application-level telemetry directly with underlying metrics. InferenceProfiler supports time-series profiling to enable continuous monitoring of the resources consumed by LLM inference.

Using InferenceProfiler, we demonstrate resource requirements for LLM inferencing on alternate GPUs by benchmarking Llama-3.2-3B-Instruct on six AWS EC2 accelerated computing instance types. With a load of 1,000 concurrent inference requests, we find that the g5.xlarge instance (A10G) achieves the lowest cost per generated token ($0.0936 per run, $0.1829 per million tokens) despite being neither the least expensive ec2 GPU instance per hour nor the fastest. Additionally, the g7e.2xlarge instance (RTX 6000 Blackwell) is the fastest but 3% more expensive per token than the g6e.xlarge and 8% more costly than the g5.xlarge.

Keywords

Large Language Models, LLM Inferencing, Performance Profiling, Observability, Energy Efficiency, Cloud Cost Analysis

ACM Reference Format:
Austin Bomhold, Xinghan Chen, Morteza Nabavinejad, and Wes Lloyd. 2026. InferenceProfiler: Enhancing the Observability of Performance, Cost, and Energy of LLM Inferencing in the Cloud. In *The 20th ACM International Conference on Distributed and Event-based Systems (DEBS '26), June 23–26, 2026, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3809481.3816482



This work is licensed under a Creative Commons Attribution 4.0 International License.
DEBS '26, Lisbon, Portugal
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2693-4/2026/06
https://doi.org/10.1145/3809481.3816482

1 Introduction

Hosting Large Language Model (LLM) inference on cloud-based GPU virtual machines is becoming the dominant deployment pattern for production AI workloads. Industry analysts forecast hyperscale data-center market growth from $209B in 2024 to $725B by 2030, with the largest cloud providers projected to spend over $600B on AI-targeted capital expenditures in 2026 alone [1, 18]. While training workloads have historically attracted the largest individual deployments, growth in inferencing workloads are forecast to grow roughly 79% annually through 2030 outpacing training which is forecast to grow 25% annually. As such, inference is projected to consume the majority of AI infrastructure by the end of the decade [9]. For a cloud consumer, LLM inferencing workloads can run across a plethora of GPU-equipped virtual machines.

A persistent challenge in cloud computing for cloud consumers who rely entirely on cloud infrastructure is observability. Optimizing deployment of inferencing workloads for cost or energy requires visibility into how these workloads exercise GPUs, virtual host servers, and inference engines, but observability for each of these components can vary and be insufficient. A key challenge for monitoring inferencing workloads is that each system component features different monitoring tools and libraries. Linux, for example offers a broad range of available tools (e.g. htop, perf).

Resource monitoring for NVIDIA GPUs is exposed through the NVIDIA Management Library (NVML) for direct queries and NVIDIA Nsight Systems (nsys) for heavier kernel tracing, with nvidia-smi and nvtop providing CLI front-ends over the same APIs. Alternative accelerators expose their own telemetry interfaces — AMD GPUs feature the AMD System Management Interface (AMD SMI) library, AWS Inferentia instances expose neuron-monitor, while Google's Tensor Processing Units (TPUs) offer the TPU Monitoring Library (libTPU SDK). Proprietary accelerator monitoring solutions can report different metrics, units, and granularity. Inference engines used to host LLMs and manage scheduling of inferencing workloads, also expose their own metrics endpoints, but these report specific engine-level state without the system context needed to attribute resources.

This observability gap matters because instance selection or ongoing request routing in heterogeneous deployments has direct economic consequences. Available GPU-enabled cloud virtual machines feature roughly an order of magnitude in hourly price differences across instance generations. Without monitoring, this broad range of virtual server alternatives offers unclear performance characteristics in serving inferencing workloads. Developers making cloud deployment or capacity-planning decisions can greatly benefit from cost-performance metrics such as cost-per-token, and observability of concurrency-headroom on GPUs. These provide key performance metrics for inferencing workloads beyond simple vendor-published peak FLOP/s. In heterogeneous clusters with multiple accelerator types, cost-per-token provides a key comparison metric, since it is derived entirely from workload output and instance pricing. Researchers and practitioners providing infrastructure for serverless inference or elastic GPU allocation need support for open, reproducible measurements of infrastructure that works across commodity cloud instances.

This paper introduces the InferenceProfiler, an open-source resource profiler for cloud-hosted LLM inference. InferenceProfiler integrates five server-side collectors to monitor and aggregate: VM metrics (CPU, memory, disk, network), cgroup counters, Linux process metrics, NVIDIA NVML GPU telemetry, and inference engine metrics (via a Prometheus endpoint) — to produce time-aligned monitoring samples. The InferenceProfiler supports continuous time-series profiling at a configurable sampling interval, as well as point-in-time resource utilization snapshots. We apply the InferenceProfiler to benchmark vLLM serving Llama-3.2-3B-Instruct across six AWS GPU instance types spanning five NVIDIA GPUs under concurrent and sequential workload profiles.

By addressing the LLM inferencing monitoring gap, the InferenceProfile aims to unlock two practical capabilities. First, availability of empirical cost-per-token tables separated by workload profile becomes the basis for instance-selection heuristics that vendor specs cannot provide. Second, time-aligned resource traces of system, cgroup, GPU, and inference-engine metrics can enable practitioners and researchers to better diagnose where workload bottlenecks exist — at the host, in the engine's scheduler, or in the GPU's memory hierarchy — enabled by data from the InferenceProfiler. This paper provides the following contributions:

- We present InferenceProfiler, an open-source profiling tool that collects host, cgroup, process, NVIDIA GPU, and vLLM metrics into a single time-aligned record.
- We report on the results for an initial empirical measurement study applying InferenceProfiler to evaluate concurrent and sequential LLM inference across six AWS EC2 GPU instance types spanning five NVIDIA architectures, reporting cost-per-token, latency distributions, and energy efficiency.

2 Background

2.1 LLM Inference Engines

LLM inference engines provide specialized software to support hosting and executing LLMs across GPU-enabled servers. Inference engines manage request scheduling, VRAM optimization, and continuous batching to maximize throughput across available hardware while minimizing latency. Research and development of LLM inference engines has grown rapidly. A recent survey catalogs 25 open-source and commercial engines with varying optimization targets, hardware support, and deployment models [15]. Among the most widely adopted are vLLM, llama.cpp, and Ollama. vLLM is a high-performance serving library whose PagedAttention mechanism stores the KV cache in fixed-size blocks, enabling dynamic sharing across requests and efficient handling of variable-length sequences [7]. At serving time, vLLM can use continuous batching to schedule prefill and decode iterations together. Llama.cpp is a lightweight C++ runtime with broad hardware portability, supporting inference across CPUs and GPUs with minimal external dependencies and multiple quantization formats[8]. Ollama wraps llama.cpp in a user-facing Go server primarily targeting single-GPU local deployments while providing a one-line installation script and supporting model management[12]. Other notable engines include NVIDIA's TensorRT-LLM, which targets maximum throughput through low-level kernel optimizations and is NVIDIA-exclusive LLM inference engine. HuggingFace Text Generation Inference (TGI) is an inference engine optimized for production deployments of common LLMs such as Llama. SGLang is another engine which emphasizes structured generation and flexible prompt execution.

For our initial design of the InferenceProfiler we chose vLLM for three reasons. First, its PagedAttention memory manager stores the KV cache in fixed-size blocks, enabling dynamic sharing across requests and efficient handling of variable concurrency. Second, vLLM uses continuous batching to schedule prefill and decode iterations together, which is the dominant serving-mode we aim to characterize. Third, vLLM provides a built-in benchmarking client with deterministic prompt generation and configurable concurrency, which simplifies reproducible cross-instance comparisons.

A persistent gap in LLM inference tooling is the disconnect between application-level observability and hardware-level resource accounting. The OpenTelemetry project has begun defining vendor-neutral semantic conventions for generative AI telemetry, standardizing attribute names for model, token count, and latency across providers [14]. Tools such as OpenLIT build on this foundation, offering client-side instrumentation that traces LLM API calls with token counts, request latency, and per-call cost attribution, alongside per-device GPU metrics [13]. While OpenLIT provides a convenient starting point for API-level cost tracking, its focus limits GPU telemetry and does not capture the metrics needed to attribute performance bottlenecks on inference servers. Both Pytorch and Nvidia have dedicated profilers that integrate with the vLLM inference engine. However, both are for offline profiling and would be at least disruptive if not prohibitive to production serving.

2.2 LLM Performance Metrics

The InferenceProfiler provides four LLM inferencing specific performance metrics. To define these metrics, let t_{req}, t_1, and t_N denote the wall-clock timestamps of request arrival, first emitted token, and final emitted token, and let N be the number of output tokens.

- **Time to first token (TTFT)** is $t_1 - t_{\text{req}}$, captures the latency of the prefill phase.
- **Inter-token latency (ITL)** is $t_i - t_{i-1}$ for $i > 1$, represents the per-step (token) decode latency.
- **End-to-end latency (E2E)** is $t_N - t_{\text{req}}$ and decomposes as $\text{TTFT} + \sum_{i=2}^{N} \text{ITL}_i$.

- **Tokens per second (TPS)** is $N/(t_N - t_1)$ and captures token throughput by calculating token count over the duration.

3 Related Work

3.1 Performance Evaluation of LLM Inference

Several prior studies have focused on benchmarking LLM inference across hardware platforms. Reddo et al. developed a reusable performance benchmark known as MLPerf Inference. The MLPerf Inference benchmark is considered the most broadly adopted ML inference benchmark, contributing standardized load generation, latency-bounded throughput scenarios, and quality targets across more than 30 organizations, though its original workloads predate large language models [17]. Samsi et al. conducted one of the earliest energy-focused inference studies, measuring Llama models on NVIDIA V100 and A100 GPUs with multi-node sharding across up to 32 GPUs, but their evaluation was limited to two GPU generations without cost analysis [19]. LLM-Inference-Bench provides an open-source benchmarking framework to evaluate LLM performance across diverse AI accelerators and inference engines. Venkata et al. benchmarked models from the Llama, Mistral, and Qwen families on a mix of NVIDIA, AMD, Habana, and SambaNova hardware using multiple inference frameworks, but their evaluation focused on offline batch throughput rather than online serving latency [5]. Emani et al. evaluated transformer workloads across diverse AI accelerators using a three-tier approach—a core transformer-block microbenchmark, a GPT-2 model, and a science use case (GenSLM)—but their focus was on training rather than serving [6]. CAR-LLM provides a machine-learning-based recommender for cloud accelerator instance selection, but targets offline inference only and does not evaluate online serving with concurrent requests [2]. None of these prior works evaluates the full range of NVIDIA cloud GPU instance types available on a single cloud provider under online serving with concurrent requests, nor evaluates dollar-per-token as a cross-instance comparison metric.

3.2 LLM Inference on Public Cloud Platforms

A growing body of work addresses LLM inference specifically on public cloud infrastructure, with the majority proposing scheduling or placement algorithms that consume per-instance performance data as input. SpotServe dynamically reparallelizes LLM serving across preemptible GPU VMs, using stateful inference recovery to cheaply resume generation upon spot preemption while reducing P99 tail latency by up to 9.1 times compared to existing systems [11]. SageServe characterizes Microsoft Office 365 Copilot workloads across Azure regions and proposes forecast-aware auto-scaling using ILP-based GPU VM allocation, reporting 25% GPU-hour savings on 10M+ daily requests [10]. A cost-efficient VM selection framework for cloud-based LLM inference optimizes KV-cache offloading based on SLOs and workload characteristics, reporting up to 73.7% cost-efficiency improvement for online workloads [4]. LLM-PILOT is an SLO-aware serving system for public cloud VM clusters that uses offloading to achieve cost-efficient LLM serving while meeting latency targets [3]. POLCA characterizes LLM inference power consumption in cloud GPU clusters and enables 30% higher server density through power oversubscription with minimal performance loss [16]. Our work is complementary: where these efforts propose scheduling algorithms, selection frameworks, or power management strategies that require per-instance performance data, InferenceProfiler enables measurement of cloud AI/ML infrastructure to produce the required performance data.

4 Methodology

4.1 InferenceProfiler

In this section, we describe the design and implementation of the InferenceProfiler, a Go-based sampling profiler that runs as a sidecar process alongside the inference server with either a CLI or HTTP server for interaction. The InferenceProfiler integrates five collectors to produce a single time-aligned record: VM-level metrics (CPU, memory, disk, network) read from `/proc` and `/sys`; cgroup counters supporting both v1 and v2; Linux process metrics; per-device NVIDIA GPU telemetry via NVML through the `go-nvml` bindings; and vLLM's Prometheus endpoint. Combined, the InferenceProfiler collects and aggregates 156 total metrics. Table 1 summarizes the collectors and representative metrics.

For each metric, the InferenceProfiler records a nanosecond-precision metric collection timestamp rather than a single per-tick timestamp, eliminating time skew across sources with different collection costs. Additionally, each collector operates in its own dedicated goroutine which provides an independent lightweight thread to prevent slower collectors from creating a bottleneck. The profiler can operate faster than the longest running collector by reading from an internal cache and relying on the per-metric timestamps for actual rates. While most profiling operations require negligible time, some, like pcieRx and pcieTx which must sample over a blocking 20ms time period, are slower. We tested that the InferenceProfiler can support a 100ms sampling interval without relying on caching and as low as a 10ms sampling interval without failing to collect samples at the rate requested. We configured a 500ms sampling rate throughout this study. Metrics collected from a single profiling run are identified using a session UUID, enabling clean cross-source joins at analysis time. The profiler appends each tick's observations to a single JSON file over the course of the run. Individual collectors can be enabled/disabled independently via command-line flags.

Profiler testing was performed on a g4dn.xlarge by running 50 concurrent LLM inference requests (n=500) at four sampling intervals (1, 10, 100, 1000 ms) compared against the workload with no profiling. Tests used the same model, hardware, request schedule, and token counts. Figure 1 shows the duration each individual collector ran to capture requested profiling data based on the sampling interval.

Each duration shown represents the total time required to obtain all metrics for the respective collectors. At a 1 ms sampling interval, the vLLM, Process, and Nvidia collectors fail to keep up. Across all conditions, mean end-to-end latency falls within ±0.14% of baseline showing that the collector imposes minimal overhead on the workload. Table 3 reports end-to-end latency for each condition.

After an initial profiling run, we reduced the full set of metrics by removing constant gauges, all-zero counters, and metrics that were constant across runs. The retained metrics span GPU thermals, SM clocks, memory utilization, PCIe throughput, and power draw from the NVML collector; CPU mode times, context switches, load average, and page faults from the VM collector; and KV-cache

Table 1: InferenceProfiler collectors and representative metrics.

Collector	Description	Representative Metrics
VM (cpu)	Host processor and general system information	loadAvg, userTime, kernelTime, idleTime, contextSwitches
VM (mem)	Host memory usage details	memFree, memCached, memBuffers, pgFault, majorPgFault
VM (disk)	Block device I/O counters	sectorReads, sectorWrites, ioTime
VM (net)	Per-interface network traffic	bytesSent, bytesRecvd, packets, errors, drops
cgroup	process group metrics (CPU, memory, I/O) from /sys/fs/cgroup	cpuTime, memoryUsed, diskIO, netIO
process	process level state and resource utilization (CPU, memory, I/O)	cpuTime, rss, virtualMem, threads
GPU	NVIDIA device telemetry via NVML	smClocks, memUtil, powerEnergy, temperature, pcieRx/Tx
vLLM	inference engine state via Prometheus	kvCacheUsage, numRequestsRunning/Waiting, ttftHist, e2eLatencyHist

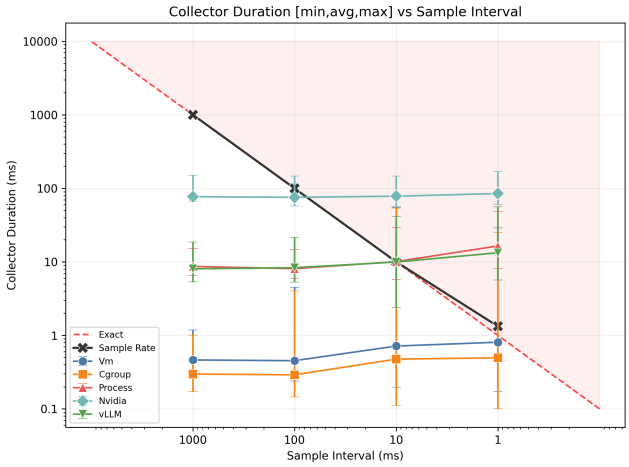


Figure 1: Profiling Times Based on Sampling Interval

Table 2: End-to-end request latency (ms) across profiler polling intervals.

Interval	Min	Max	Mean	Δ
1 ms	31230.8	40217.0	35909.6	+0.11%
10 ms	31213.2	40219.1	35820.5	−0.14%
100 ms	31005.4	40218.4	35853.6	−0.05%
1000 ms	31145.1	40108.9	35830.2	−0.12%
Baseline	31183.0	40175.9	35871.5	0.00%

occupancy, running/waiting request counts, and latency histograms from the vLLM collector.

4.2 Experimental Setup

Table 3 summarizes the six AWS GPU instances evaluated. All instances used in our experiments were EBS-backed and located in us-east-2. We calculate costs using on-demand ec2 instance prices observed in April 2026. Our instance selection spans four generations of NVIDIA GPU silicon: Turing (T4), Ampere (A10G), Ada Lovelace (L4, L40S), and Blackwell (RTX PRO 6000). These instances feature on-demand hourly pricing spanning 6.4x vs. the minimum cost, slightly less than an order of magnitude.

Table 3: AWS GPU instances evaluated.

Instance	GPU	VRAM	vCPU	$/hr
g4dn.xlarge	T4	16 GB	4	0.5260
g4dn.2xlarge	T4	16 GB	8	0.7520
g5.xlarge	A10G	24 GB	4	1.0060
g6.xlarge	L4	24 GB	4	0.8048
g6e.xlarge	L40S	48 GB	4	1.8610
g7e.2xlarge	RTX PRO 6000	96 GB	8	3.3631

The model under test is Meta's Llama-3.2-3B-Instruct with 3.21B parameters in BFloat16, which requires 6.4 GB of static VRAM for model weights. vLLM is run with chunked prefill to batch the prefill tokens alongside the decoding phase. The vLLM version and runtime environment are identical across all instances.

Load is generated by vLLM's built-in benchmarking client running on a separate EC2 instance, t4g.xlarge, in the same availability zone. Requests are issued against vLLM's OpenAI compatible HTTP endpoint. Prompts are drawn from vLLM's deterministic random-prompt generator with a fixed seed and 0 model temperature, so the exact same prompt stream is replayed across all instance types. We report on our profiling of two distinct workloads.

- Workload W1 (concurrent) issues 1,000 requests at 50-way concurrency with 512 random tokens for both input and output, exercising the continuous-batching scheduler under sustained load.
- Workload W2 (sequential) issues 100 requests at concurrency 1 on the g6e.xlarge (L40S), with input and output lengths independently varied across 256, 512, and 1,024 tokens around a 512-token baseline, isolating the effect of token length on per-request latency.

5 Results

5.1 Concurrent Workload: Throughput and Cost

For a workload of wall-clock duration Δt seconds on an instance with hourly on-demand price p generating N output tokens, the cost per million output tokens is $(p \cdot \Delta t \cdot 10^6)/(N \cdot 3600)$. Table 4 reports per-instance results for workload W1.

The most cost-effective per-token instance (g5.xlarge) is neither the least expensive per hour nor the fastest. While the g4dn.xlarge cost 1.9 times less per hour, it provided 2.2 times lower throughput which more than consumed the savings, yielding 15% higher

Table 4: Workload W1: 1,000 requests, 512-token input/output, 50-way concurrency.

Instance	GPU	Time (s)	TPS	$/M output token
g4dn.xlarge	T4	736.19	695.5	0.2101
g4dn.2xlarge	T4	719.26	711.8	0.2934
g5.xlarge	A10G	335.14	1,527.7	**0.1829**
g6.xlarge	L4	549.88	931.1	0.2401
g6e.xlarge	L40S	190.53	2,687.2	0.1924
g7e.2xlarge	RTX PRO 6000	**108.58**	**4,715.6**	0.1981

costs per million output tokens processed. The g7e.2xlarge was the fastest instance offering 6.8x the throughput compared to the g4dn.xlarge's T4 GPU. The g7e.2xlarge's 6.4x higher hourly price however, resulted in a 3% higher cost-per-token compared to the g6e.xlarge (and 8% higher than the g5.xlarge). Doubling the vCPU count on the T4 GPU (g4dn.xlarge to g4dn.2xlarge) yielded only a 2% throughput increase at a 40% cost increase, consistent with the workload being GPU-bound rather than CPU-bound at this model size.

5.2 Concurrent Workload: Latency Distributions

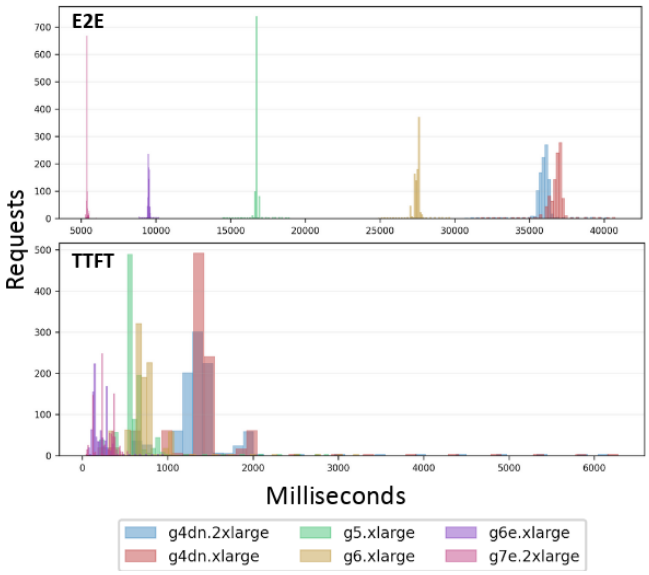


Figure 2: End-to-End Latency & Time to First Token

Figure 2 shows the per-request latency distributions under workload W1. End-to-end latencies separate cleanly across instances: approximately 5s for the g7e, 9.5s for the g6e, 16.5s for the g5, 27.5s for the g6, and 36–37s for the two g4dn instances, with limited overlap between adjacent tiers. The g4dn instances exhibit visibly broader distributions with larger tails than higher-end instances, spanning roughly double the interquartile range of the next closest instance (g6), indicating greater variance under sustained concurrent load.

TTFT distributions concentrate in three bands: 100–500ms for the g7e and g6e, 500–1,000ms for the g5 and g6, and 1,000–2,000ms for the g4dn variants. All six instances exhibit a long TTFT tail extending beyond the primary mode, most pronounced on the T4 instances.

5.3 Sequential Workload: Token Length Sensitivity

Table 5 reports results from workload W2 on the g6e.xlarge (L40S), where input and output lengths were varied independently around a 512-token baseline at concurrency 1.

Table 5: Workload W2: 100 sequential requests on g6e.xlarge (L40S).

In	Out	E2E (s)	TTFT (ms)	TPS	$/M out
256	512	4.91	21.5	**104.3**	**4.957**
512	256	**2.46**	22.5	103.9	4.973
512	512	4.92	19.8	104.0	4.972
512	1024	9.91	**19.4**	103.3	5.005
1024	512	5.00	31.8	102.3	5.048

Output length was the dominant driver of per-request latency observed in our sequential workload test. Doubling output tokens from 256 to 512 doubles E2E from 2.46s to 4.92s, and doubling again to 1,024 doubles it to 9.91s, confirming that decode time scales linearly with output length when no batching is present. Output TPS remains nearly constant across all five configurations (102–104 tokens/s), indicating that the L40S sustains a stable per-token decode rate regardless of sequence length at this model scale.

Input length has a smaller but measurable effect. Doubling input tokens from 256 to 1,024 increases mean TTFT from 21.5ms to 31.8ms as the prefill phase processes a longer prompt, but adds only 0.09s to total E2E since the additional prefill cost is amortized over the 512-token decode.

Comparing W1 and W2 on the same hardware illustrates the effect of continuous batching. Under W1's 50-way concurrency the g6e.xlarge sustained 2,687 output tokens per second at $0.192 per million output tokens, while under W2's sequential execution TPS drops to 104 and cost per million output tokens rises to $4.97, an increase of approximately 26x, demonstrating that concurrent batching is a primary lever for cost-efficient serving at this model scale.

5.4 Energy Efficiency

InferenceProfiler's NVML collector records cumulative GPU energy consumption via the `PowerEnergy` counter, enabling per-run energy attribution. For each instance under workload W1, we compute kilowatt-hours per million output tokens as $(\Delta E \cdot 10^6)/(N \cdot 3.6 \times 10^6)$, where ΔE is the energy delta in joules over the run and N is the total output tokens. Figure 3 plots kWh per million output tokens against cost per million output tokens for all six instances.

No single instance dominated both cost and energy efficiency. The g5.xlarge (A10G) achieved the lowest cost per million output tokens ($0.183) but the highest energy per token (35.7 kWh/M). The g7e.2xlarge (RTX PRO 6000) drew the least energy per token

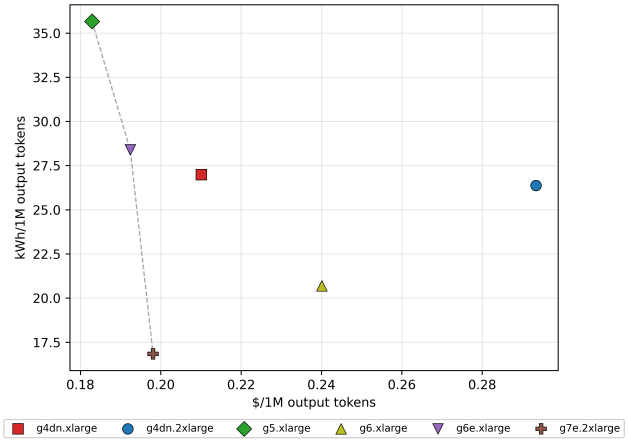


Figure 3: kWh per Million Tokens vs Cost per Million Tokens

(16.8 kWh/M) at a near-best $0.198 per million. The g6e.xlarge (L40S) sat between them at $0.192 per million and 28.4 kWh/M. The g4dn.xlarge, g4dn.2xlarge, and g6.xlarge were dominated on both axes. The g7e.2xlarge and g6.xlarge (L4) emerge as the most energy-efficient on a per-token basis, primarily through higher throughput that reduces the wall-clock time the GPU is powered, even though their instantaneous power draw is higher.

These results highlight that cost-per-token and energy-per-token are complementary but non-identical optimization targets. Additionally, specific Nvidia GPU energy measurements should translate to other clouds and environments having the same GPUs. Cost-sensitive batch workloads may favor lower-priced instances despite higher energy consumption, while energy-constrained or sustainability-focused deployments benefit from newer GPU generations that complete work faster for a lower total energy cost.

6 Conclusion

This paper presented the InferenceProfiler, an open-source profiler that unifies VM, cgroup, process, NVIDIA GPU, and inference-engine telemetry into a single time-aligned record to characterize cloud-hosted LLM inference. We applied the InferenceProfiler to benchmark vLLM serving Llama 3.2-3B across six AWS GPU instance types spanning five NVIDIA architectures under concurrent and sequential workloads.

Three findings emerge from the measurement study. First, the most cost-efficient per token instance (g5.xlarge / A10G at $0.183 per million output tokens) is neither the least expensive per hour nor the fastest, demonstrating that instance selection based on hourly price or vendor-published peak performance leads to sub-optimal cost efficiency. Second, no single instance dominates both cost and energy efficiency, indicating that these are complementary optimization targets that require workload-specific measurement rather than static heuristics. Third, continuous batching at 50-way concurrency reduces cost per million output tokens by approximately 26 times compared to sequential execution on the same hardware, confirming that batching is the primary lever for cost-efficient serving at this model scale.

Our results illustrate the value of integrated, multi-layer profiling for LLM inference on heterogeneous cloud GPUs. Future work can expand the evaluation to larger multi-GPU models and additional inference engines such as TensorRT-LLM and llama.cpp to determine whether the observed cost and energy rankings generalize across different model scales and software stacks.

References

[1] Carbon Credits. 2026. *AI Demand to Drive $600B from the Big Five for GPU and Data Center Boom by 2026.* https://carboncredits.com/ai-demand-to-drive-600b-from-the-big-five-for-gpu-and-data-center-boom-by-2026/ Accessed: 2026-04-27.

[2] A. Krishnan et al. 2024. CAR-LLM: Cloud Accelerator Recommender for Large Language Models. In *IEEE Int. Conf. on High Performance Computing, Data, and Analytics (HiPC)*. 89–99. doi:10.1109/HiPC62374.2024.00018

[3] J. O. Lazo Tembra et al. 2024. LLM-PILOT: Characterize and Optimize Performance of your LLM Inference Services. In *Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*. doi:10.1109/SC41406.2024.00065

[4] K. Kim et al. 2025. Cost-Efficient VM Selection for Cloud-Based LLM Inference with KV Cache Offloading. In *IEEE Int. Conf. on Cloud Computing (CLOUD)*. doi:10.1109/CLOUD62652.2025.00029

[5] K. T. Chitty-Venkata et al. 2024. LLM-Inference-Bench: Inference Benchmarking of Large Language Models on AI Accelerators. In *SC24-W: Workshops of the Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. 1362–1379. doi:10.1109/SCW63240.2024.00178

[6] M. Emani et al. 2024. Toward a Holistic Performance Evaluation of Large Language Models Across Diverse AI Accelerators. In *IEEE Int. Parallel and Distributed Processing Symp. Workshops (IPDPSW)*. 1–10. doi:10.1109/IPDPSW63119.2024.00016

[7] W. Kwon et al. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proc. 29th Symp. on Operating Systems Principles (SOSP)*. 611–626. doi:10.1145/3600006.3613165

[8] ggml-org. 2026. *llama.cpp: LLM inference in C/C++.* Accessed: 2026-05-01.

[9] Iron Mountain. 2025. *Data Centers: AI Inferencing Infrastructure.* https://resources.ironmountain.com/whitepapers/d/data-centers-ai-inferencing-infrastructure Accessed: 2026-04-27.

[10] Atal Soni et al. Jaiswal. 2025. SageServe: Proactive Service Level Agreement Aware Autoscaling of Machine Learning Serving Systems. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 9, 3 (2025). doi:10.1145/3720283

[11] Xupeng Miao, Chunan Shi, Jiangfei Duan, Xiaoli Xi, Dahua Lin, Bin Cui, and Zhihao Jia. 2024. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2.* doi:10.1145/3620666.3651383

[12] Ollama. 2026. *Ollama: Get up and running with LLMs.* Accessed: 2026-05-01.

[13] OpenLIT. 2024. *OpenLIT: Open Source Observability for AI.* https://github.com/openlit/openlit Accessed: 2026-04-27.

[14] OpenTelemetry. 2024. *Semantic Conventions for Generative AI.* https://opentelemetry.io/docs/specs/semconv/gen-ai/ Accessed: 2026-04-27.

[15] Sihyeong Park, Sungryeol Jeon, Chaelyn Lee, Seokhun Jeon, Byung-Soo Kim, and Jemin Lee. 2026. A Survey on Inference Engines for Large Language Models: Perspectives on Optimization and Efficiency. *ACM Transactions on Intelligent Systems and Technology* (2026). doi:10.1145/3803798

[16] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Aashaka Shah, Saeed Maleki, and Ricardo Bianchini. 2024. Characterizing Power Management Opportunities for LLMs in the Cloud. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3.* doi:10.1145/3620666.3651329

[17] Vijay Janapa et al. Reddi. 2020. MLPerf Inference Benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 446–459. doi:10.1109/ISCA45697.2020.00045

[18] ResearchAndMarkets.com. 2025. *Hyperscale Data Centers Business Research Report 2025: Market to Reach $724.9 Billion by 2030 at a CAGR of 23%.* https://www.businesswire.com/news/home/20251106341686/en/ Accessed: 2026-04-27.

[19] Siddharth Samsi et al. 2023. From Words to Watts: Benchmarking the Energy Costs of Large Language Model Inference. In *2023 IEEE High Performance Extreme Computing Conference (HPEC)*. doi:10.1109/HPEC58863.2023.10363447