Evaluating Energy Efficiency and Performance in LLM Inference with DheyoGPUpd

Shaswot Paudel*
Lava Bokam†, Abhilash Ghanore†
Aakash Varma Nadimpalli‡

*Main contributor †Main mentors ‡Additional contributor

shaswotp@dheyo.ai lava@dheyo.ai
abhilash@dheyo.ai aakashv@dheyo.ai

February 5, 2026

Abstract. LLM serving is increasingly constrained by energy, not just latency and throughput, motivating efficiency metrics that reflect the full cost of autoregressive decoding. We evaluate inference trade-offs for Meta's LLaMA-3.1-8B under a controlled decoding workload (single-token inputs generating 512 tokens) while sweeping batch size and GPU clock frequency. We report Tokens per Joule (TPJ), tokens per second (TPS), end-to-end latency, and time-between-tokens (TBT), and identify configurations that favor (i) energy-efficient serving (large batches at moderate clocks), (ii) latency-sensitive streaming (small batches at higher clocks), and (iii) throughput-driven operation (large batches at high clocks). Finally, we introduce DheyoGPUpd: a lightweight, unified telemetry and GPU power-control agent for heterogeneous NVIDIA+AMD fleets that exposes Prometheus metrics, supports push-based MQTT publishing, and enables real-time policy-driven power/clock management.

1 Introduction

Large Language Models (LLMs) are increasingly integrated into production systems across industries. These models require significant computational resources while serving, especially when deployed at scale.

According to the International Energy Agency (IEA), global data center electricity consumption reached approximately 460 TWh in 2022, accounting for about 1.5% of global electricity demand. This is projected to rise significantly by 2026, with AI workloads, especially large-scale inference, being a key contributor.[1, 2]

In today's AI/ML clusters, HPC workloads, and media processing environments, workloads increasingly run across heterogeneous accelerator fleets that combine NVIDIA and AMD GPUs within the same data center and, in some cases, even within the same node. This trend is driven by supply constraints, cost–performance trade-offs, and workload specialization.

For tasks such as bottleneck detection, dynamic tuning, and preventing thermal or hardware failures, a robust telemetry system is essential. Telemetry, however, is more than just a performance tool—it plays a critical role in energy management.

Prior work has highlighted both the scale of data-center energy use and the uncertainty/variability in global energy-use estimates, motivating careful measurement and efficiency-focused system design.[3]

The urgency of energy-efficient computing is also reflected in the growing literature on the environmental and economic costs of modern deep learning workloads.[4, 5]

Just as fuel efficiency became a defining factor in the automotive industry, energy efficiency in compute infrastructure will soon be a central concern. Controlling GPU power consumption is therefore a critical step toward mitigating the massive and growing energy demands of modern compute workloads.

For data center operators, this trend introduces challenges related to infrastructure scaling, operational expenditure, and sustainability. LLM inference in particular involves complex trade-offs between latency, throughput, and power consumption. As energy costs rise, evaluating models not only by throughput and latency but also by efficiency metrics such as energy per request and tokens per joule (TPJ) becomes increasingly important.

This study analyzes how batch size and GPU frequency impact the energy efficiency and performance of LLM inference using Meta's Llama family of models (Llama 3.1–8B).[6]

2 The Challenge of Energy-Efficient LLM Serving

Inference with large language models (LLMs) introduces distinctive challenges due to their autoregressive decoding behavior, where tokens are generated sequentially. Unlike conventional inference, the computational workload during LLM decoding evolves over time and depends on parameters like batch size and decoding length.

This study focuses specifically on the decoding stage of inference. Each input consists of a single token, and the model is tasked with generating 512 output tokens. This controlled setup allows consistent evaluation of compute load and power draw across different runs while excluding variability introduced by prompt length.

Traditional power management techniques such as static power capping are often suboptimal for LLM workloads. A central metric in this study is Tokens per Joule (TPJ), defined as the number of generated tokens divided by the energy consumed during decoding. TPJ captures both throughput and power efficiency, making it a useful composite indicator of overall inference cost-effectiveness. Since energy is computed as power integrated over time, TPJ also implicitly accounts for how fast tokens are generated and how much power the system uses during that time.

Understanding how TPJ responds to different batch sizes and GPU frequencies is critical for identifying configurations that deliver high generation throughput while minimizing energy expenditure.

While Tokens per Joule (TPJ) is the primary metric for assessing energy efficiency, it does not capture all relevant performance trade-offs. Depending on the deployment objective—whether minimizing power costs, reducing response time, or maximizing throughput—different metrics become more appropriate.

3 Challenges in Heterogeneous GPU Environments

In heterogeneous GPU environments, telemetry itself becomes a major operational challenge:

- **Vendor-specific telemetry stacks:** NVIDIA, AMD, and other accelerators each expose different telemetry interfaces (e.g., DCGM, ROCm SMI), requiring separate exporters for each vendor.
- **Inconsistent metric schemas:** Different exporters use divergent naming conventions and labeling for similar metrics, complicating PromQL queries, alerting rules, and cross-vendor comparisons.
- **Lack of node-level integration:** GPU telemetry is often isolated from system-level signals (CPU, memory, PCIe bandwidth, power draw), hindering root-cause analysis for performance regressions or resource contention.
- **Limited data delivery models:** Most exporters rely solely on pull-based scraping, limiting flexibility for real-time or push-driven telemetry use cases.
- **Operational overhead at scale:** Large deployments must manage, configure, and update multiple exporters per node type, plus separate system-level exporters, leading to higher DaemonSet counts, complex scrape configurations, and increased change management burden, especially in Kubernetes and managed Prometheus environments.
- **Lack of integrated control capabilities:** Existing systems are primarily read-only; they provide metrics but cannot proactively adjust GPU parameters (e.g., power limits, clock speeds) in response to performance or energy considerations.

4 Metrics and Evaluation Goals

This study reports multiple performance and efficiency metrics to guide configuration selection based on three common serving goals. These modes represent trade-offs. Understanding the relationship between batch size, GPU frequency, and each metric helps guide configuration choices depending on the specific LLM use case.

4.1 Energy-Efficient Serving

Key metrics: Tokens per Joule (TPJ), Energy per Request, Average Power Consumption.

This mode is suitable for cost-sensitive LLM deployments, where minimizing energy use is a priority. In such cases, response time is less critical, so settings that optimize for energy per token are preferred (e.g., lowering electricity and cooling costs in data centers).

4.2 Latency-Sensitive Serving

Key metrics: End-to-End Latency, Time Between Tokens (TBT).

This mode applies to interactive/streaming applications, where fast output tokens are needed to maintain responsiveness. Low latency ensures fluid back-and-forth interaction (e.g., conversational AI chatbots where users expect near real-time token streaming).

4.3 Throughput-Driven Serving

Key metric: Tokens per Second (TPS).

This mode is relevant for high-throughput inference pipelines, where the objective is to process large volumes efficiently. Here maximizing tokens per second becomes critical, even if latency or power usage is higher (e.g., LLM inference behind an API for large-scale users such as coding assistants or enterprise SaaS tools with many parallel requests).

5 Results and Discussion

5.1 Energy Efficiency Analysis

As shown in Figure 1, Tokens per Joule (TPJ) improves with both batch size and GPU frequency, peaking around Batch 32 at 1000–1400 MHz. Energy per request drops significantly with larger batches, making them more efficient overall. However, average power also increases as batch size and frequency rise, which may limit how aggressively these settings can be used in practice. From the trend, large batch size at moderate GPU frequencies (1000–1400 MHz) offers the best energy efficiency while keeping power draw manageable.

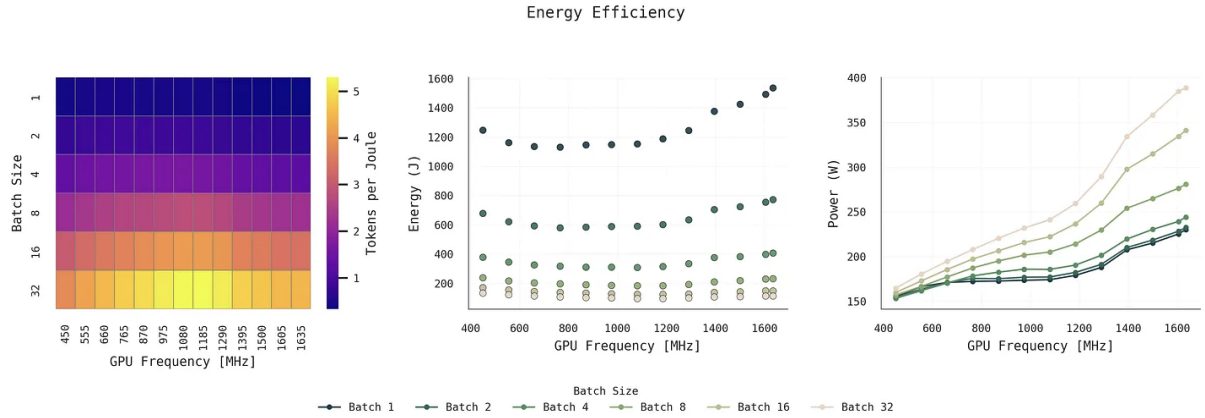

Figure 1: Energy Efficiency: Energy metrics across batch sizes and GPU frequencies, showing trade-offs in tokens per joule, energy per request, and power consumption.

5.2 Latency Analysis

As shown in Figure 2, latency decreases with smaller batch sizes and higher GPU frequencies. Batch 1 consistently shows the lowest Time Between Tokens (TBT) and end-to-end latency across all settings. Larger batches incur higher latency, which can be mitigated but not eliminated by

increasing the frequency. For batch sizes 1 and 2, latency is already low, and increasing GPU frequency beyond 700–800 MHz offers minimal improvement, making these lower frequencies a good trade-off point for reduced power consumption.

For applications that prioritize fast response time, using Batch 1 or 2 with high GPU frequencies ($\geq$ 1200 MHz) is the most effective setup.

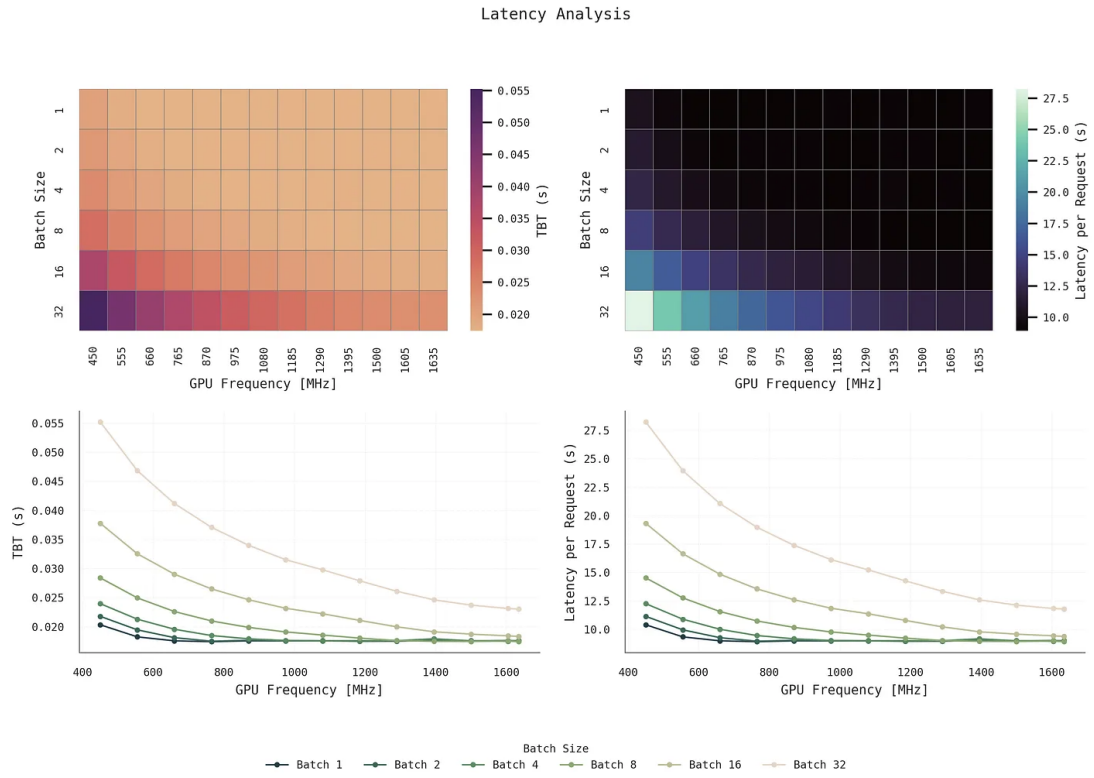

Figure 2: Latency Analysis: Impact of batch size and GPU frequency on Time Between Tokens and end-to-end latency for low-latency applications.

5.3 Throughput Analysis

As shown in Figure 3, throughput increases steadily with both batch size and frequency. Batch 32 around 1600 MHz achieves the highest tokens-per-second rate. Even at lower frequencies, larger batches outperform smaller ones in throughput.

For throughput-focused scenarios, larger batch sizes at high GPU frequencies (1400–1600 MHz) are the most suitable configuration.

5.4 Summary of Trade-offs

Note: These observations are based on the LLaMA 3.1–8B model. While exact values may vary across models, the overall trends with respect to batch size and GPU frequency are expected to remain consistent. The goal is to understand these trends so they can guide configuration choices

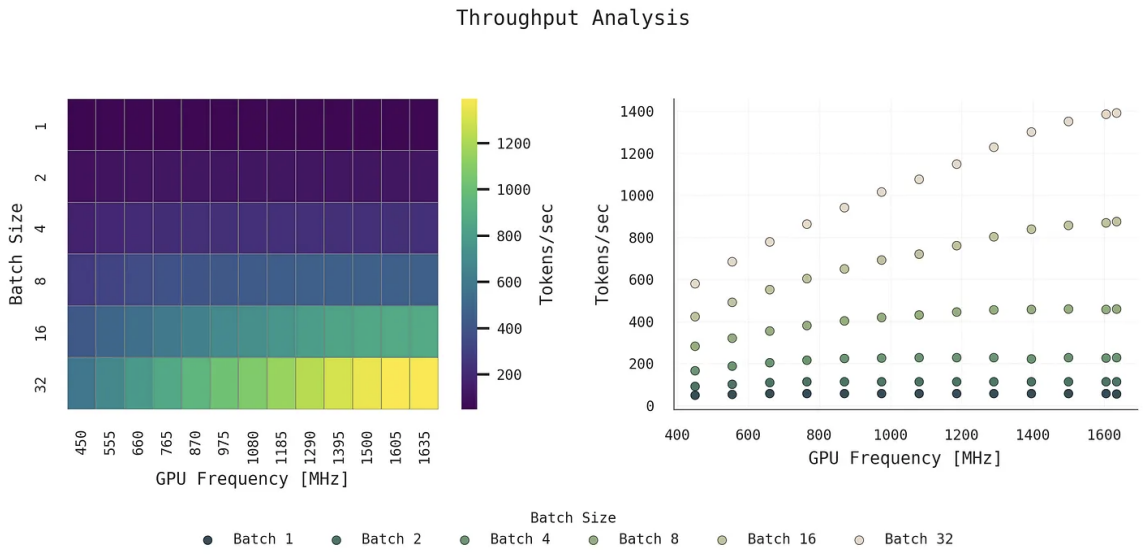

Figure 3: Throughput Analysis: Throughput (tokens/sec) scales with batch size and GPU frequency, as shown by heatmap and scatter plot.

for different LLMs.

Table 1: Recommended configurations by serving goal (LLaMA 3.1–8B).

Serving goal	**Best batch size**	**Best GPU freq**	**Notes**
Energy efficiency	32	1000–1400 MHz	High TPJ and low energy per request; power draw is reasonable.
Low latency	1 or 2	$\geq$ 700 MHz	For batch sizes 1 and 2, latency remains low even at 700–800 MHz, making them optimal for low-power, low-latency serving.
High throughput	32	1400–1600 MHz	Maximizes tokens/sec; suitable for high-load scenarios.

6 Introducing DheyoGPUpd

DheyoGPUpd is a robust and efficient solution for monitoring and controlling GPU resources in real-time.

6.1 Features

- **Comprehensive Monitoring:** Full support for NVIDIA and AMD GPUs, along with system-level and CPU metrics for complete observability.
- **Dual Control Plane:** Actively manage GPU performance parameters, such as clock speeds, via either a REST API or an MQTT message bus.

- **Flexible Metrics Delivery:** Supports push-based metric publishing over MQTT for event-driven or real-time use cases, and pull-based scraping via a native `/metrics` endpoint for seamless Prometheus integration.
- **High Performance:** Designed with a concurrent, asynchronous architecture using Rust's `tokio` runtime and a multi-threaded C++ backend for efficient AMD metrics collection.
- **Memory Safety:** Core application logic is implemented in Rust, ensuring reliability and eliminating common memory-related vulnerabilities.
- **Container-Ready:** Includes a production-ready Dockerfile and Kubernetes DaemonSet configuration for easy deployment in containerized and cloud environments.
- **Graceful Degradation:** Automatically detects the absence of GPUs and continues to operate by providing system-only metrics, ensuring monitoring continuity.

6.2 Comparison

Table 2: Comparison of common telemetry exporters vs. DheyoGPUpd.

Exporter	Size	System metrics	NVIDIA metrics	AMD metrics
Node Exporter	14.6 MB	✓	✗	✗
DCGM Exporter	75 MB	✗	✓	✗
AMD SMI Exporter	12 MB	✗	✗	✓
DheyoGPUpd	18 MB	✓	✓	✓

6.3 System Overview

GPUpd consists of two main components: the Telemetry Module and the Power Control Module (Figure 4).

6.3.1 Telemetry Module

The Telemetry Module is responsible for real-time collection, aggregation, and forwarding of GPU and system metrics. It ensures high availability and flexibility for monitoring in both standalone and distributed GPU environments.

Key Features

- **Multi-vendor support:** Collects metrics from NVIDIA, AMD, and system GPUs in parallel.
- **Flexible data export:**
  - **Prometheus:** Exposes metrics via `/metrics` endpoint for scraping.
  - **MQTT:** Publishes metrics to the `dheyo/metrics` topic (configurable broker endpoint).

- **Alerting & diagnostics:**
  - Built-in alerting system for threshold-based notifications.
  - Differentiates between hardware-level faults and system-level errors.
- **Scalability:** Designed for real-time monitoring across single or multiple GPUs in a cluster.

6.3.2 Power Control Module

The Power Control Module enables fine-grained GPU power and performance management, giving administrators precise control over GPU behavior.

Key Features

- **Clock speed management:** Dynamically tune GPU clock frequencies to optimize performance or reduce power consumption.
- **Power caps:** Define maximum power limits for GPUs to balance performance and energy efficiency.
- **Scheduling:** Automate power and clock adjustments based on workloads, time-of-day, or policies.
- **Granular scope:** Apply control policies to individual GPUs or clusters of GPUs simultaneously.
- **Flexible control methods:**
  - Publish control commands to the `dheyo/power-control` topic via MQTT.
  - Send JSON-based requests to the `/power-control` HTTP endpoint.

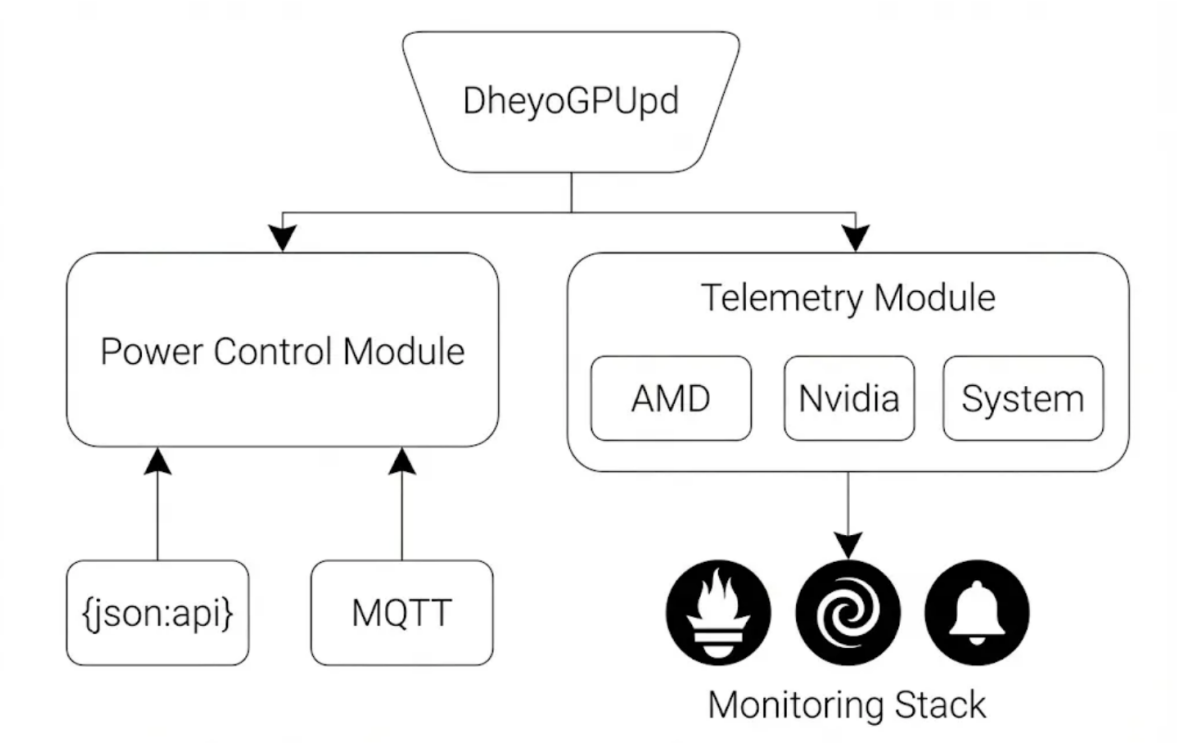

Figure 4: High Level Design.

7 Conclusion

DheyoGPUpd unifies GPU telemetry and power control across heterogeneous environments, offering a single lightweight solution for monitoring and managing NVIDIA, AMD, and system-level resources. By combining real-time observability with fine-grained control, it simplifies operations, improves efficiency, and helps reduce the growing energy footprint of modern compute workloads.

References

[1] International Energy Agency (IEA). *Data Centres, Artificial Intelligence and Cryptocurrencies*. IEA report/analysis, 2024.

[2] Data Center Dynamics. *Global data center electricity use to double by 2026 – IEA report*. News article, 2024.

[3] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, Jonathan Koomey, and others. "Recalibrating global data center energy-use estimates." *Science*, 367(6481):984–986, 2020.

[4] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Deep Learning in NLP." In *Proceedings of ACL*, 2019.

[5] Roy Schwartz, Jesse Dodge, Noah A. Smith, and Oren Etzioni. "Green AI." *arXiv preprint arXiv:1907.10597*, 2019.

[6] Abhishek Dubey and the Llama 3 team. "The Llama 3 Herd of Models." *arXiv preprint arXiv:2407.21783*, 2024.

[7] Woosuk Kwon, Zhuohan Li, Ioannis Antonoglou, and others. "vLLM: Easy, Fast, and Cheap LLM Serving with PagedAttention." *arXiv preprint arXiv:2309.06180*, 2023.

[8] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. "FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness." *arXiv preprint arXiv:2205.14135*, 2022.

[9] Yonatan Leviathan, Matan Kalman, and Yossi Matias. "Fast Inference from Transformers via Speculative Decoding." *arXiv preprint arXiv:2302.01318*, 2023.

[10] Guangxuan Xiao, Ji Lin, Song Han, and others. "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models." *arXiv preprint arXiv:2211.10438*, 2022.