Technical Report 2026-01-01
Internetworking and Media Communications Research Laboratories
Department of Computer Science, Kent State University
http://medianet.kent.edu/technicalreports.html

Workload Characterization of Midsize GPU LLM Inference Under High-Concurrency Batching

Sharmila Rahman Prithula and Javed I. Khan

Media Communications and Networking Research Lab
Department of Computer Science
Kent State University, Kent OH 44242
javed | sprithul@kent.edu

Abstract: *Large Language Models (LLMs) are increasingly deployed on self-hosted mid-sized inference platforms where aggressive batching and multi-tenant concurrency are required to maximize hardware utilization. Such midsize infrastructures are becoming increasingly important for sovereign and institutionally autonomous AI deployment because they reduce dependence on external inference providers while enabling local operational control. In this paper, we present a workload-aware characterization of concurrent LLM inference using vLLM and Text Generation Inference on a midsize NVIDIA A40 platform across multiple model scales and concurrency regimes. From detailed measurements of GPU utilization, power draw, and latency, we observe a scheduler bottleneck inversion: for small, parameter-efficient models, increasing concurrency shifts the primary bottleneck away from the GPU and onto the host CPU. The GPU becomes underutilized while the host processor struggles to keep up with frequent scheduling decisions, meaning that smaller models do not always simplify deployment and can instead overload the CPU. Furthermore, we found that aggressive batching dramatically improves overall hardware efficiency and reduces median latency. Based on these findings, we argue that future LLM serving systems should move beyond throughput-centric optimization toward workload-aware and quality-sensitive scheduling strategies that jointly consider GPU utilization, host-side overhead, energy efficiency, and application-level quality of service. This paper presents a workload aware characterization of concurrent LLM inference that jointly identifies a scheduler bottleneck inversion where smaller models shift pressure from GPU to host CPU, and reveals an emergent trade-off between hardware efficiency under aggressive batching.*

1. Introduction

Large Language Models (LLMs) are increasingly being deployed on midsize, self-hosted inference platforms. These infrastructures are vital for organizations requiring data sovereignty, operational autonomy, and reduced reliance on external API providers. However, maximizing hardware utilization on constrained single-GPU or midsize platforms requires aggressive batching and multi-tenant concurrency. While modern inference engines like vLLM and Text Generation Inference (TGI) have introduced

Cite this document as: Sharmila Rahman Prithula and Javed I. Khan and, *Workload Characterization of Midsize GPU LLM Inference Under High-Concurrency Batching*, Technical Report 2026-01-01, Internetworking and Media Communications Research Laboratories, Department of Computer Science, Kent State University, January 2026, Available at: http://medianet.kent.edu/technicalreports.html

optimizations such as PagedAttention and continuous batching to improve aggregate throughput, current evaluation paradigms remain heavily throughput-centric. These standard metrics often fail to capture how diverse, heterogeneous workloads and varying model scales impact system-level bottlenecks under sustained load. Furthermore, while lightweight models are frequently deployed to conserve GPU memory, their operational behavior under high concurrency is not fully understood. In this paper, we present a workload-aware characterization of concurrent LLM inference using a self-hosted NVIDIA A40 platform. We evaluate two state-of-the-art serving engines (vLLM and TGI) across three model scales (Llama-1B, Mistral-7B, and Llama-8B) using the heterogeneous LEWIS-60 benchmark suite. Our characterization yields below primary contributions:

1. We identify a scheduler bottleneck inversion specific to parameter-efficient models (sub-5B), where aggressive concurrency shifts the primary system limitation away from GPU compute saturation and onto the host CPU scheduler.

2. We demonstrate that aggressive batching dramatically improves overall hardware efficiency and energy-delay product (EDP), though it introduces persistent tail-latency variability.

2. Background and Hardware Context

LLM Inference Runtimes: Recent years have witnessed substantial advances in Large Language Model (LLM) inference infrastructure, particularly in the areas of continuous batching, memory-efficient serving, distributed inference, and hardware-aware runtime optimization. Modern inference systems primarily aim to maximize throughput and accelerator utilization while minimizing latency and memory fragmentation [1, 2, 3].

Continuous batching and dynamic scheduling systems such as Orca introduced iteration-level scheduling to improve utilization and reduce head-of-line blocking in transformer inference workloads [3]. Subsequent systems including vLLM further improved inference efficiency through PagedAttention-based KV-cache virtualization, significantly reducing memory fragmentation and increasing achievable request concurrency [1]. Similarly, distributed and phase-aware serving systems such as DeepSpeed-Inference [5], Splitwise [9], AlpaServe [17], and SpotServe [18] explored tensor parallelism, pipeline partitioning, statistical multiplexing, and elastic resource allocation to improve large-scale LLM serving efficiency.

A parallel line of research has focused on hardware-aware optimization of transformer inference. Studies on GPU microarchitectural behavior, memory bandwidth utilization, FlashAttention, and inference scaling laws have improved understanding of transformer execution efficiency on modern accelerators [6, 7, 16]. Benchmarking initiatives such as MLPerf Inference standardized throughput and latency evaluation across heterogeneous hardware platforms [11]. Additional work on quantization and model compression, including GPTQ [10] and AWQ [15], further explored reducing computational and memory overhead for deployment on constrained infrastructure.

Hardware and Workload Characterization: Despite these advances, most existing studies primarily evaluate inference systems using throughput-centric metrics such as tokens-per-second, average latency, or aggregate accelerator utilization [2,5,11]. In many cases, inference workloads are treated as architecturally homogeneous token streams with limited consideration of how workload diversity may alter scheduler dynamics, queue behavior, memory pressure, or batching efficiency under sustained multi-tenant concurrency.

Recent work has observed that CPU scheduling overhead can dominate inference time [8], motivating optimizations like multi-step scheduling and even CPU-free architectures. However, these efforts treat CPU overhead as a general problem without recognizing that it becomes the bottleneck specifically for smaller models under concurrency, a regime shift we characterize as scheduler bottleneck inversion. Concurrently, EDP has emerged as a metric for energy-aware serving, but existing work does not examine the trade-off between hardware efficiency and output quality. Our paper bridges these gaps by providing the first systematic characterization of how small-model concurrency and heterogeneous workloads jointly affect both infrastructure behavior and application-level QoS. Rather than proposing a new inference runtime or scheduling algorithm, we focus on workload-aware infrastructure characterization of concurrent LLM serving on a self-hosted single-GPU platform. Our objective is to understand how batching concurrency alters runtime behavior at the system level under realistic heterogeneous workloads. Unlike prior throughput-centric evaluations, our study explicitly incorporates a heterogeneous benchmark suite (LEWIS-60) containing factual, logical, mathematical, reasoning, and programming-oriented prompts intended to exercise diverse inference behaviors. Finally, our work emphasizes the emerging tension between hardware efficiency and application-level reasoning utility in self-hosted inference environments. Rather than evaluating throughput in isolation, we argue that future LLM serving systems should jointly consider batching efficiency, latency stability, scheduler behavior, and workload-sensitive Quality of Service (QoS) constraints.

3. Methodology

We design a dedicated data collection pipeline that isolates active inference execution from inactive periods, allowing precise hardware-level characterization under controlled concurrency. The methodology comprises the experimental platform, inference engines, workload benchmark, model selection, metrics, and the taxonomy of workload-specific stress expected from prompt diversity.

3.1. Infrastructure and Hardware Telemetry

All benchmarks were executed on a **midsize-GPU** HPC node equipped with an NVIDIA A40 GPU (48 GB GDDR6 VRAM, PCIe Gen4) and a high-performance host CPU infrastructure. To minimize interference from co-located processes common in shared clusters, the Docker containers hosting the inference engines were pinned to dedicated CPU cores. Hardware sensor data was captured using NVIDIA's Management Library (NVML) and Data Center GPU Manager (DCGM). We monitored the GPU at sub-second

intervals to detect transient increases in active compute usage (%), VRAM allocation (GB), and GPU board-level power consumption (Watts). Host-side metrics: aggregate CPU utilization and per-core usage, were logged in parallel at 1 Hz using two standard Linux profiling utilities: mpstat (multiprocessor statistics) and pidstat (process ID statistics) [9]. Crucially, our analysis script filters out idle tail-ends: only samples where GPU compute utilization exceeded 1 % are included in averaged active-window metrics, ensuring that reported values strictly represent inference work rather than idle periods.

3.2. Inference Engines and LEWIS-60 Workload

We evaluated two state-of-the-art AI-as-a-Service runtimes:

- **vLLM (PagedAttention):** A Python-based inference server that pre-allocates VRAM into pages to mitigate KV-cache fragmentation [1].
- **Text Generation Inference (TGI):** A Rust-based router maintained by Hugging Face, optimized for continuous batching and lower host-side overhead [4].

The workload was driven by LEWIS-60, a campus-oriented benchmark suite designed to exercise diverse reasoning and decoding behaviors. For the experiments reported here, we used the full multi-category collection, which spans two broad categories: deterministic prompts (factual retrieval, logical deduction, mathematical problem solving, commonsense reasoning, and cognitive puzzles) and open-ended prompts (complex reasoning, programming tasks, descriptive generation, harm refusal, and neutrality-sensitive controversy topics). This diversity ensures that the generated tokens require varied degrees of algorithmic fidelity, enabling evaluation of application-level Quality of Service (QoS) degradation under hardware stress [12]. Concurrency C was defined as the number of simultaneous in-flight requests. We varied C from 1 (baseline single-user execution) to 24 (maximum multi-tenant stress). Requests were issued via the engines' standard HTTP APIs using an asynchronous client that always maintained exactly C outstanding requests. When one request was completed, the client immediately injected the next prompt from a shuffled queue, keeping the system in a steady-state backlog. This approach reflects continuous high-load conditions encountered in shared institutional deployments.

3.3. Model Selection and Parameter Scaling

To evaluate how architectural weight impacts silicon efficiency [6], we deployed three instruction-tuned models:

1) Llama-3.2-1B-Instruct: A highly parameter-efficient model to test host-side scheduling limits under extreme concurrency.

2) Mistral-7B-Instruct-v0.3: A dense model featuring Grouped-Query Attention (GQA), representative of popular open-weight medium-capacity LLMs [13].

3) Llama-3.1-8B-Instruct: A larger baseline for evaluating throughput-latency behavior at higher parameter counts [14].

These models were chosen to span a parameter range from extremely light (1B) to moderately heavy (8B), enabling the study of both GPU compute saturation and the host-side bottleneck inversion that occurs when per-token work is minimal. All models were loaded with automatic data-type detection (--dtype auto) and GPU memory utilization capped at 0.9 of available VRAM. Key deployment parameters are summarized in Table 1.

Table 1: Key Deployment Parameters

Engine	Model	Key Parameters
vLLM	Llama-3.2-1B-Instruct Llama-3.1-8B-Instruct Mistral-7B-Instruct-v0.3	--max-model-len 8192 --dtype auto --gpu-memory-utilization 0.9 --ipc=host
TGI	Llama-3.2-1B-Instruct Llama-3.1-8B-Instruct Mistral-7B-Instruct-v0.3	--shm-size 1g (4g for Mistral) Continuous Batching Enabled

3.4. Workload Stress Taxonomy

To contextualize the measured system behavior, we classify each prompt category in LEWIS-60 according to its expected impact on KV-cache growth, GPU memory pressure, and scheduler/batching dynamics. This taxonomy is based on prompt length, typical output length, and decoder behavior observed in preliminary profiling. It serves as a qualitative reference for interpreting the heterogeneous stress signatures we report. (Note: the empirical validation of these patterns is presented in Section 4.)

3.5. Composite Metrics

Standard throughput (tokens/second) fails to capture the energetic cost of queueing delays and tail latency. Therefore, we calculate the Energy-Delay Product (EDP), which jointly penalizes high power draw and long response times:

$$EDP = (\text{Average Power} \times \text{Average Latency}) \times \text{Average Latency}$$

Here Average Latency is the mean end-to-end request latency (in seconds) for a given concurrency level, and Average GPU Power (Watts) is measured over the active inference window. The resulting unit is Joules × seconds; lower EDP indicates a more energy-proportional and responsive system.

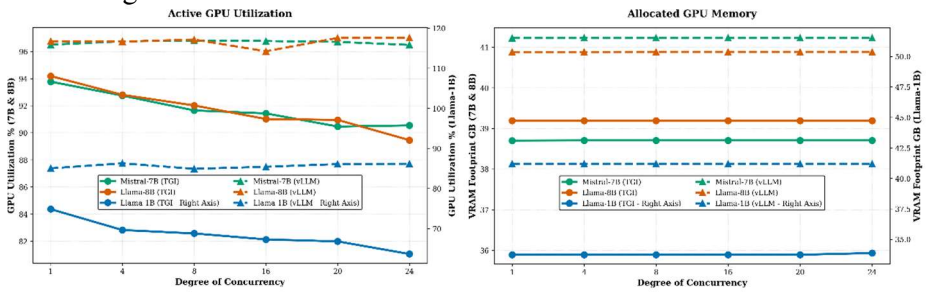
4. Hardware Characterization and Evaluation

Across all following figures, colors denote the specific model architecture, while line styles distinguish the inference engine (solid lines with circular markers for TGI; dashed lines with triangular markers for vLLM); additionally, to prevent scale compression, metrics for

the parameter-efficient Llama-1B model are consistently plotted on a secondary right-hand y-axis.

4.1. GPU Utilization and Resource Occupancy

Our measurements reveal clear differences in how model scale and serving backend influence active GPU utilization under sustained concurrency. As shown in Figure 1(a), the larger Llama-8B and Mistral-7B models rapidly approach near-saturation of the A40 GPU as concurrency increases, whereas the lightweight Llama-1B model remains substantially below full utilization even at maximum concurrency. This behavior indicates that parameter-efficient models are unable to fully occupy the accelerator despite aggressive batching.



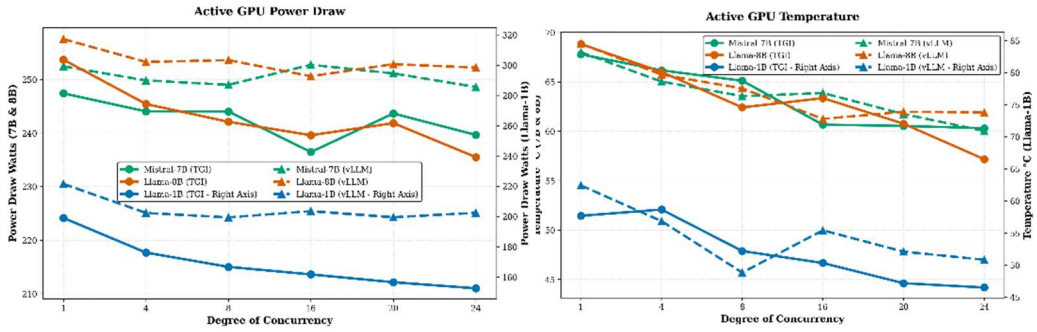
(a) (b)

Figure 1: (a) Active GPU Utilization, (b) Allocated GPU Memory footprint under increasing concurrency. Llama-1B (secondary axis) fails to saturate the SMs despite maximum concurrency.

However, an independent samples t-test shows that vLLM consistently maintains higher hardware occupancy than TGI. Across all runs, vLLM achieved a mean active GPU utilization of 95.05% compared to TGI's 90.32% ($t = -21.50$, $p < 0.0001$). The difference is not only statistically significant but also practically meaningful: the Cohen's d value of −0.619 indicates a medium-to-large effect size – roughly 0.6 standard deviations separate the two engines' utilization means. Furthermore, a two-way factorial ANOVA evaluated how GPU power draw depends on both the model scale (e.g., 1B vs. 8B) and the serving backend (vLLM vs. TGI). The analysis revealed a highly significant interaction between the two factors ($F = 158.4$, $p < 0.0001$). This is most clearly seen in the lightweight Llama-1B model, where switching from TGI to vLLM increased power by roughly 30 W (183.9 W vs. 213.7 W). In contrast, for the 8B model the same backend switch changed power by only 6.7 W. The effect size of this interaction is given by partial $\eta^2 = 0.062$, meaning that about 6.2% of the total variance in GPU power is explained specifically by the *interaction* – a small but statistically solid contribution. As further supported by the VRAM allocations in Figure 1(b), these results indicate that for models with fewer than 5 billion parameters (sub-5B), the choice of memory allocator (PagedAttention in vLLM vs. TGI's approach) significantly influences the hardware's power profile and utilization ceiling – long before the GPU's computational capacity is exhausted.

4.2. Power Draw and Thermal Dynamics

Sustained concurrent inference produces substantial power and thermal load on the A40 platform. Figure 2(a) shows that the larger Llama-8B and Mistral-7B models consistently operate near the GPU's board-level power envelope, with sustained consumption approaching 250 W under high concurrency. Correspondingly, Figure 2(b) shows elevated operating temperatures for the larger models, with temperatures frequently approaching 70 °C during prolonged inference execution. Across all configurations, vLLM exhibits slightly higher average power draw than TGI. One possible explanation is the additional runtime overhead associated with maintaining PagedAttention metadata structures and more aggressive batching behavior, although additional profiling would be required to isolate the precise source of the difference. Interestingly, concurrency scaling exhibits a statistically significant negative correlation with GPU temperature (Spearman $\rho = -0.538$, $p < 0.0001$). As concurrency increases from C = 1 to C = 24, average GPU temperature decreases from 67.4 °C to 56.8 °C.



(a) (b)

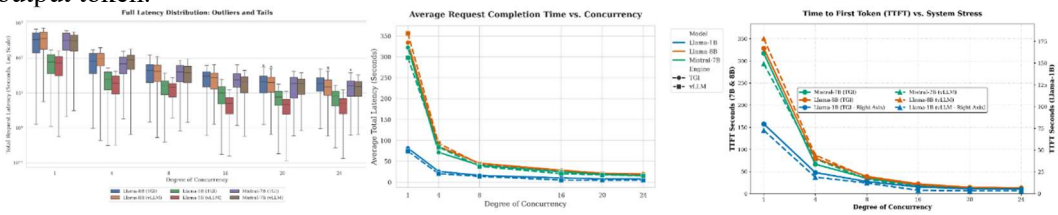
Figure 2: (a) Active GPU Power Draw, (b) Thermal Dynamics. Larger models consistently push the A40 toward its 250W power limit and trigger thermal throttling under load.

Similarly, GPU power also decreased with higher concurrency, though the correlation was weaker: $\rho = -0.284$ ($p < 0.0001$) – a small but statistically significant negative relationship. Why would higher concurrency lead to lower temperatures and power? One plausible explanation is that aggressive continuous batching allows the GPU to stay busy with useful memory-bound operations, rather than sitting in a high-power idle state waiting for a single batch's queue to clear. In other words, batching improves operational efficiency, reducing the amount of time the GPU spends in inefficient, power-hungry idle loops.

4.3. Queueing Dynamics and Tail Latency

Our measurements show that concurrency-driven batching substantially reshapes request-level latency behavior. As concurrency increases, the inference engines become more effective at maintaining continuously filled execution queues, significantly reducing request completion time while simultaneously increasing host-side scheduling activity. Figure 3(a) shows the evolution of tail-latency distributions across concurrency regimes,

while Figure 3(b) illustrates the corresponding reduction in average request completion time. Figure 3(c) further characterizes interactive responsiveness through Time-to-First-Token (TTFT), capturing the delay between request submission and generation of the first output token.



(a) (b) (c)

Figure 3: (a) Tail Latency Distributions (log scale), (b) Average Request Completion Time. (c) Time-to-First-Token (TTFT) behavior under sustained multi-tenant batching. Increasing concurrency substantially compresses median latency and improves responsiveness, although persistent tail variability and scheduler induced TTFT overhead remain visible under aggressive batching.

Table 2: Latency Distribution and Predictability (Llama-8B)

Engine	Concurrency	Median / p50 (s)	Tail / p95 (s)	Extreme / p99 (s)
TGI	C = 1	335.04	642.65	662.92
	C = 4	79.62	166.06	169.39
	C = 8	44.05	90.39	100.24
	C = 16	29.79	50.70	54.91
	C = 20	20.79	44.10	59.71
	C = 24	18.32	39.07	46.96
vLLM	C = 1	360.72	668.77	724.04
	C = 4	96.17	164.10	169.40
	C = 8	41.31	91.19	102.91
	C = 16	26.72	46.27	56.06
	C = 20	19.84	38.30	49.26
	C = 24	14.83	33.65	47.97

Table 2 presents the latency distribution for the Llama-8B model across all evaluated concurrency levels. Under single-request execution (C = 1), both runtimes exhibit extremely high median latency, reaching 335.04 s for TGI and 360.72 s for vLLM. These results indicate that low-concurrency execution leaves the GPU underutilized while requests incur prolonged queue residence time due to insufficient batching efficiency. As concurrency increases, batching becomes significantly more effective. Median latency decreases progressively across both runtimes, reaching 18.32 s for TGI and 14.83 s for vLLM at C = 24. Tail latency follows a similar trend, with p95 and p99 latency decreasing sharply as concurrency improves queue occupancy and amortizes execution overhead across larger request batches.

However, latency variability remains persistent even after batching efficiency improves. For vLLM, the coefficient variation rises from 0.600 at C = 1 to 0.669 at C = 8 before stabilizing at 0.633 at C = 24, remaining above the single-user baseline. This suggests that

while aggressive batching substantially reduces average queueing delay, asynchronous scheduling dynamics and heterogeneous decode lengths continue to introduce tail instability under high concurrency. Figure 3(c) shows that TTFT follows a more nuanced trend than end-to-end latency. Moderate concurrency improves TTFT through more efficient batch formation, but then gains plateau at higher concurrency, particularly for lightweight models where requests complete rapidly and runtime scheduling frequency increases substantially. This effect is especially visible for Llama-1B, suggesting that host-side orchestration overhead increasingly influences responsiveness as model execution time decreases. These results demonstrate that concurrency-driven batching effectively alleviates underfilled execution regimes and improves throughput efficiency, but does not completely eliminate tail-latency variability or scheduling-induced responsiveness overhead under sustained multi-tenant load.

Table 3: Hardware Sensor Data and Resource Footprint (Mean Across All Concurrency Levels)

Configuration	GPU Util (%)	GPU Power (W)	GPU Temp (°C)	CPU Util (%)	VRAM Allocated (MB)
Model Architectures					
Llama-3.1-8B	95.03%	252.45	66.69	16.15%	5,580
Mistral-7B	94.25%	247.14	65.98	16.22%	5,323
Llama-3.2-1B	77.84%	197.28	57.04	18.90%	5,460
Serving Backends					
TGI	90.32%	239.05	64.92	16.08%	5,758
vLLM	95.05%	248.55	65.61	16.93%	5,167

4.4. The Scheduling Bottleneck Inversion

Contrasting the lightweight Llama-1B model against the larger 7B and 8B architectures reveals a notable shift in system bottlenecks under sustained concurrency. Across the full dataset, Spearman rank correlations show that increasing concurrency produces a statistically significant increase in host CPU utilization ($\rho = +0.420$, $p < 0.0001$), while GPU utilization exhibits a slight negative correlation with concurrency ($\rho = -0.125$, $p < 0.0001$). Table 3 shows that Llama-1B imposes consistently higher host CPU utilization than either Mistral-7B or Llama-8B across all concurrency regimes. Specifically, Llama-1B requires 18.90% mean CPU utilization compared to 16.22% for Mistral-7B and 16.15% for Llama-8B. At the same time, the lightweight model exhibits substantially lower GPU utilization and power draw. This behavior suggests a bottleneck inversion effect. Because the GPU executes the smaller model extremely rapidly, requests complete at a higher frequency, forcing the host runtime to perform scheduling, batching, and queue management operations more frequently to maintain concurrency. A Mann-Whitney U test on per-request token generation rates further confirms that Llama-1B generates tokens significantly faster than the larger models ($p < 0.001$). Consequently, the primary system limitation shifts away from accelerator-side compute saturation toward host-side orchestration overhead. These findings indicate that aggressively parameter-efficient models may not proportionally reduce infrastructure stress; instead, they redistribute pressure toward the CPU scheduler and runtime management layer under high-concurrency serving conditions.

4.5. Architectural Efficiency Through Energy-Delay Product

To evaluate overall hardware efficiency beyond throughput alone, we computed the Energy-Delay Product (EDP), which jointly captures energy consumption and latency behavior. Table 4 summarizes the EDP measurements for Llama-8B across all concurrency levels. Single-request execution exhibits extremely poor efficiency due to prolonged latency combined with sustained GPU power draw. For example, vLLM at C = 1 produces an EDP of 32,446,460 Joule · Seconds, reflecting the combined penalty of high latency and continuous power consumption during underutilized execution. As concurrency increases, EDP decreases dramatically for both runtimes.

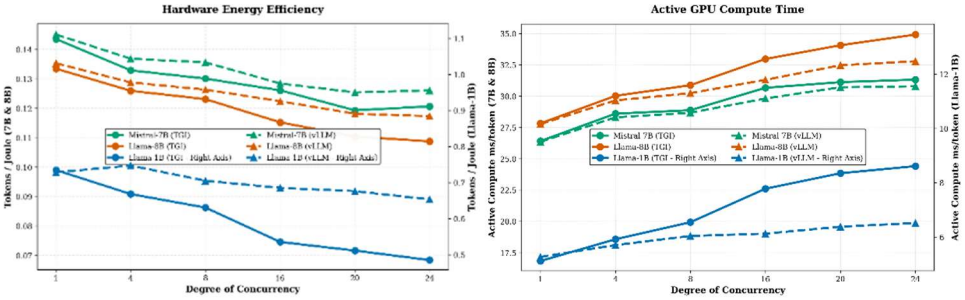
The largest improvement occurs between C = 1 and C = 8, indicating that batching substantially improves energy proportionality by reducing idle waiting and increasing useful GPU work per unit time. Beyond C = 16, however, efficiency gains begin to diminish, suggesting that additional concurrency provides progressively smaller benefits once batching overheads and scheduling costs become dominant. Among all evaluated configurations, vLLM achieves its lowest EDP at C = 24 (69,108 Joule · Seconds), indicating the most favorable balance between latency and power efficiency under sustained load. These observations demonstrate that concurrency-driven batching fundamentally alters the energy efficiency characteristics of LLM inference systems.

Table 4: Energy-Delay Product (EDP) measurements for Llama-8B across concurrency regimes

Engine	Concurrency	Mean Power (W)	Mean Latency (s)	EDP (Joules · Seconds)
TGI	C = 1	249.0	333.95	27,769,128
	C = 4	249.0	85.34	1,813,445
	C = 8	249.0	45.24	509,617
	C = 16	249.0	28.57	203,244
	C = 20	249.0	21.17	111,594
	C = 24	249.0	19.85	98,111
vLLM	C = 1	255.7	356.22	32,446,460
	C = 4	255.7	92.94	2,208,696
	C = 8	255.7	43.44	482,514
	C = 16	255.7	25.83	170,600
	C = 20	255.7	20.02	102,484
	C = 24	255.7	16.44	69,108

4.6. Energy Efficiency and Token-Level Cost

To further evaluate serving efficiency, we measured the energy required per generated token across all concurrency regimes. Figure 4(a) shows that token-level efficiency improves sharply as concurrency increases from C = 1 to moderate batching levels. At low concurrency, the larger models exhibit extremely high energy cost per token because static GPU power dominates the relatively small amount of useful computational work being performed. As batching improves between C = 1 and C = 8, energy efficiency increases substantially and the Joules-per-token curve flattens progressively.



(a) (b)

Figure 4: (a) Hardware Energy Efficiency in Tokens per Joule, (b) Active GPU Compute Time per Token. Note the asymptotic efficiency curve as concurrency scales.

Nevertheless, the larger models remain significantly less energy-efficient than the lightweight Llama-1B configuration. Table 5 quantifies this difference at maximum concurrency (C = 24). Llama-1B achieves approximately 0.49 Tokens/Joule (≈2.0 Joules/token), whereas Mistral-7B achieves only 0.12 Tokens/Joule (≈8.3 Joules/token). This substantial efficiency gap reflects the increased computational and memory requirements of larger parameter architectures. Figure 4(b) further illustrates that GPU compute time per token decreases asymptotically with concurrency, reinforcing the observation that batching improves execution efficiency most strongly at lower concurrency levels before gradually reaching diminishing returns. Taken together, these results indicate that moderate-to-high batching is essential for energy-efficient LLM serving, although increasingly aggressive concurrency eventually yields smaller incremental efficiency gains while earlier sections demonstrated continued latency variability under sustained load.

Table 5: Host Overhead: Llama-1B VS Mistral-7B at C=24

Model	Joules/Token	Tokens/Joule
Llama-1B	≈2.0	0.49
Mistral-7B	≈8.3	0.12

Table 6: Statistical Significance and Effect Sizes of Infrastructure Variables

Variable / Interaction	Metric Impacted	Test Statistic	p-value	Effect Size	Interpretation
Model Scale (1B vs 7B/8B)	GPU Power (W)	ANOVA: F = 2598.9	< 0.0001	$\eta^2 = 0.518$	Massive Effect: Model size drives 51.8 % of power variance.
Backend Choice (vLLM vs TGI)	VRAM (MB)	ANOVA: F = 21748.3	< 0.0001	$\eta^2 = 0.818$	Massive Effect: PagedAttention drastically alters memory pooling.
Backend Choice (vLLM vs TGI)	GPU Util (%)	t-test: t = −21.50	< 0.0001	Cohen's d = 0.619	Medium-Large Effect: vLLM significantly increases SM occupancy.
Model × Backend	GPU Power (W)	2-Way ANOVA: F = 158.4	< 0.0001	η^2_p = 0.062	Significant Interaction: Backends scale

					power differently based on model size.
Concurrency Scaling	CPU Util (%)	Spearman: $\rho = 0.420$	< 0.0001	–	Moderate Positive Correlation: Validates the host scheduler bottleneck.
Concurrency Scaling	GPU Temp (°C)	Spearman: $\rho = -0.538$	< 0.0001	–	Strong Negative Correlation: Batching cools the GPU via execution efficiency.

4.7. Statistical Synthesis of Infrastructure Constraints

To formalize our characterization of the hardware telemetry, we conducted a suite of parametric and non-parametric statistical tests, summarized in Table 6. Because Shapiro-Wilk tests confirmed the hardware telemetry was highly non-normal (typical for systems queues), we utilized robust effect sizes and rank correlations to validate our findings. The analysis confirms that the chosen model architecture is the absolute dominant driver of GPU power variance ($\eta^2 = 0.518$), while the backend serving engine dictates VRAM allocation behavior almost entirely ($\eta^2 = 0.818$). Furthermore, the significant interaction effect between model and backend ($F = 158.4$, $p < 0.0001$) strongly indicates that PagedAttention (vLLM) and continuous batching (TGI) scale hardware stress differently depending on the parameter weight of the model. Finally, Spearman rank correlations confirm our bottleneck inversion hypothesis: pushing concurrency reliably increases CPU scheduling strain ($\rho = 0.420$) while paradoxically dropping GPU temperatures ($\rho = -0.538$) as batching efficiency clears single-threaded queueing stalls.

5. Discussion: Implications for HPC

The results presented in Section 4 carry direct implications for the design of next-generation inference infrastructure in self-hosted, mid-scale environments.

The bottleneck inversion we documented for parameter-efficient models challenges the common assumption that smaller models always simplify deployment. With Llama-1B at C = 24, the GPU was underutilized while the host CPU became the system's limiting resource. In practice, this means that operators who select a 1B–3B model to conserve GPU memory or energy may inadvertently shift their provisioning problem from the accelerator to the host server. Future inference runtimes should therefore monitor host-side scheduling pressure explicitly and, where necessary, throttle concurrency or coalesce scheduling operations (e.g., batch multiple scheduling decisions together or reduce interrupt frequency) to avoid CPU saturation when serving very small models.

6. Conclusion

This paper presented a workload-aware characterization of concurrent LLM inference on a midsize-GPU self-hosted platform using an NVIDIA A40 running vLLM and Text Generation Inference across three model scales (Llama-1B, Mistral-7B, Llama-8B) and concurrency levels from 1 to 24, measuring GPU utilization, power draw, Energy-Delay Product, KV-cache pressure, and tail-latency distributions under the heterogeneous

LEWIS-60 benchmark suite. For parameter-efficient models (sub-5B), the system bottleneck can invert from GPU compute saturation to host-CPU scheduling overhead, exposing a previously underappreciated limitation in high-concurrency small-model serving. These results demonstrate that optimizing LLM serving purely for tokens per second leaves critical hardware efficiency and Quality of Service considerations unaddressed. Future runtime architectures should treat workload identity, host-side scheduler pressure, and algorithmic fidelity as co-equal objectives alongside latency and throughput. Our study was intentionally scoped to midsize single-GPU inference on a mid-sized campus platform which faces financial and infrastructure constraints. As future work, we plan to extend our characterization to other accelerator families (e.g., NVIDIA H100, AMD GPUs) and to larger model scales, as well as to design and evaluate workload-aware scheduling policies that react to the stress signatures we have identified. In parallel, we are currently analyzing the quality of LLM responses across the same workload categories in a companion paper, focusing on how batching and concurrency affect reasoning fidelity, factual accuracy, and output coherence. These ongoing efforts will complement the hardware-centric findings reported here.

Acknowledgement: This research and the computing infrastructure used in this work were supported by the National Science Foundation under the following awards from the Office of Cyberinfrastructure: NSF Award #2346729, Adiabatic Microservice-Level Load Balanced Forwarding in Programmable Switches for Accelerating Safe and Secure Urgent Processes in Science Data Centers; NSF Award #2201558, Accelerating Compute-Driven Science Through a Sharable High Performance Computing Cluster in the Kent State Multi-Campus System; and NSF Award #192567, A Use Case Design Through Kent State's Sharable Science DMZ. The authors also acknowledge valuable contributions and support from colleagues, especially Roy Heath, Jason Rose, Jeff Bailey, Phil Thomas, and James Raber of Kent State Research Computing

REFERENCES

[1] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*. 611–626.

[2] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Joseph E Gonzalez, and Hao Zhang. 2023. High-throughput generative inference of large language models with a single GPU. In *International Conference on Machine Learning (ICML)*. PMLR.

[3] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI '22)*. USENIX Association, 521–538.

[4] Hugging Face. 2023. Text Generation Inference. GitHub repository.12 Retrieved from https://github.com/huggingface/text-generation-inference

[5] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Awa, Jeff Reuther, and others. 2022. DeepSpeed-Inference: enabling efficient inference of transformer models at unprecedented scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–15.

[6] Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. 2023. Efficiently scaling transformer inference. In *Proceedings of Machine Learning and Systems (MLSys)*, Vol. 5.

[7] Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with io-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 35. 16344–16359.

[8] M. Siavashi, M. Scazzariello, G. Q. Maguire Jr, D. Kostić, and M. Chiesa. 2026. Blink: CPU-Free LLM Inference by Delegating the Serving Stack to GPU and SmartNIC. arXiv preprint arXiv:2604.07609.

[9] Pratyush Patel, Esha Choukse, Chaojie Zhang, and others. 2024. Splitwise: Efficient generative LLM inference using phase splitting. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA)*.

[10] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2022. GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers. arXiv preprint arXiv:2210.17323.

[11] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maxim Maximov, and others. 2020. MLPerf Inference Benchmark. In *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA)*. 446–459.

[12] Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, and others. 2022. Holistic evaluation of language models. arXiv preprint arXiv:2211.09110.

[13] Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, and others. 2023. Mistral 7B. arXiv preprint arXiv:2310.06825.

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

[15] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Xingyu Dang, and Song Han.

2024. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. In *Proceedings of Machine Learning and Systems (MLSys)*.

[16] Zhe Jia, Marco Maggioni, Benjamin Smith, and Daniele Paolo Scarpazza. 2018. Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking. arXiv preprint arXiv:1804.06826.

[17] Zhuohan Li, Lianmin Zheng, Yinmin Zhong, and others. 2023. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI '23)*.

[18] Xupeng Miao and others. 2024. SpotServe: Serving Generative Large Language Models on Preemptible Instances. In *Proceedings of the 29th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.