GreenLLM: SLO-Aware Dynamic Frequency Scaling for Energy-Efficient LLM Serving

Qunyou Liu
qunyou.liu@epfl.ch
Embedded Systems Laboratory (ESL), École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Vaud, Switzerland

Darong Huang
darong.huang@epfl.ch
Embedded Systems Laboratory (ESL), École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Vaud, Switzerland

Marina Zapater
marina.zapater@heig-vd.ch
Institute of Reconfigurable & Embedded Digital Systems (REDS), School of Engineering and Management Vaud, HES-SO University of Applied Sciences and Arts Western Switzerland
Yverdon-les-Bains, Switzerland

David Atienza
david.atienza@epfl.ch
Embedded Systems Laboratory (ESL), École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Vaud, Switzerland

Abstract

Large Language Models (LLMs) are rapidly becoming the backbone of modern cloud services, yet their inference costs are dominated by energy consumption on GPUs. Unlike traditional GPU workloads, LLM inference consists of two distinct stages with different characteristics: the prefill phase, which is latency-sensitive and scales quadratically with prompt length, and the decode phase, which progresses token by token with undetermined length. Current GPU power governors (for example, NVIDIA default) overlook this asymmetry, treating both phases uniformly. The result is mismatched voltage/frequency settings, leading to suboptimal voltage/frequency configurations, head-of-line blocking, and excessive energy consumption.

We introduce GreenLLM, a service-level objectives (SLO) aware serving framework that minimizes GPU energy by explicitly separating prefill and decode control. At ingress, requests are routed into length-based queues so short prompts avoid head-of-line blocking, tightening TTFT. For prefill, GreenLLM collects short traces on a GPU node, fits compact latency–power models over SM frequency, and solves a queueing-aware optimization to pick energy-minimal clocks per class. During decode, a lightweight dual-loop controller tracks throughput (tokens-per-second) and adjusts frequency with hysteretic, fine-grained steps to hold tail TBT within target bounds. Across Alibaba and Azure trace replays, GreenLLM achieves up to 34% reduction in total energy consumption compared to the default DVFS baseline in Alibaba/Azure trace replays, with no loss of throughput and only less than 3.5% SLO violations increase, demonstrating its effectiveness in the efficient LLM service.

Keywords: GPU, LLM, SLO, Energy Optimization, Performance, DVFS

1 Introduction

Generative Large Language Models (LLMs) have experienced explosive adoption in recent years, driving a surge in demand for GPU clusters to support inference workloads [4, 14, 15, 40]. To keep up with user requests for chatbots, virtual assistants, and other generative services, enterprises are rapidly scaling up dedicated LLM inference clusters [4–7, 14, 16, 37]. However, a central challenge in this expansion is the substantial power consumption of modern AI accelerators [46]. Recent studies estimate that OpenAI's GPT-4o consumes nearly 1.8 Wh per long query [11]. At scale, the energy are staggering: with GPT-4o projected to handle about 772 billion queries in 2025, the total electricity demand could reach 391–463 GWh, comparable to the annual consumption of 35,000 U.S. households [11].

GPUs are the primary workhorses for LLM inference, but they are also the dominant source of power consumption in serving these models. For instance, NVIDIA's H100 consumes up to 700 W per GPU [21], with high-density AI racks demanding 30–40 kW [1]. However, conventional GPU power management and scheduling techniques are often ill-suited for LLM inference. Default GPU frequency scaling governors (e.g., NVIDIA's) typically drive accelerators at high clock speeds to maximize throughput, with little regard for workload-specific characteristics or latency SLOs [35]. As shown in Fig. 1a, under a sinusoidal decode workload, NVIDIA's default governor (defaultNV) fails to adjust frequency in response to workload intensity. In contrast, Fig. 1b illustrates our proposed method, which successfully tracks the workload dynamics.

Similarly, cluster-level power and thermal management mechanisms, such as power capping or throttling, treat all GPU tasks uniformly and usually activate only in emergency conditions, resulting in suboptimal efficiency [28, 47, 51]. To address these shortcomings, researchers have explored

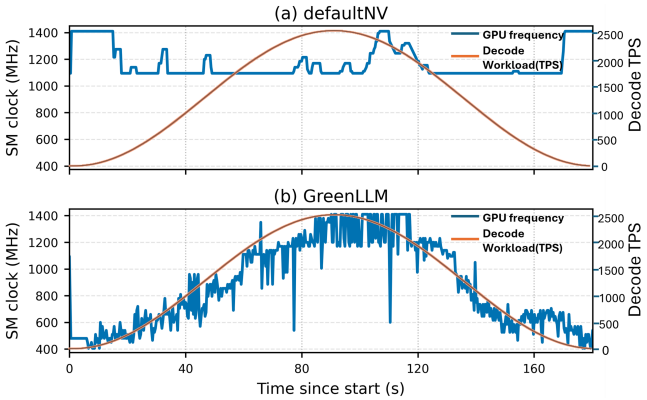


Figure 1. GPU Frequency vs. Decode TPS under defaultNV and GreenLLM

more adaptive strategies, such as fine-grained frequency scaling across different AI operators [42] and SLO-aware methods [13]. Wilkins *et al.* [43] develop offline energy models of LLM inference (parameterized by input/output tokens) to guide scheduling across heterogeneous systems, while *throttLL'eM* [12] predicts the per-query load and dynamically scales GPU frequencies, saving energy under strict latency constraints. However, these approaches operate at coarse granularity (e.g., cluster-level or offline scheduling) or depend on complex forecasting for each query.

Importantly, none explicitly exploits a fundamental property of LLM service: inference follows a two-phase execution pattern, as illustrated in Figure 2, with different characteristics [3, 27, 50], creating new opportunities for fine-grained, workload-specific power management.

In this work, we propose GreenLLM, an energy-aware LLM serving framework that applies phase-specific GPU frequency control to reduce energy consumption under strict SLO constraints. At the prefill phase, GreenLLM first classifies incoming requests into separate queues by prompt length (e.g. short vs. long prompts). This length-based routing isolates short prompts from long ones, preventing head-of-line blocking and significantly improving the time-to-first-token (TTFT) latency for short queries. Then, for each prompt-length class, we profile the LLM's prefill execution on an NVIDIA DGX-A100 (8× A100 GPUs) server and fit a compact latency–power model as a function of the GPU Streaming Multiprocessor (SM) frequency. Using this model, at runtime, GreenLLM can solve a queueing-aware optimization problem to select the optimal SM frequency to execute prefill phases, maximizing energy efficiency while ensuring incoming requests meet latency SLO.

During the decode phase, GreenLLM employs a lightweight dual-loop feedback controller that dynamically adjusts the GPU frequency in real time. Every 20 ms, the controller measures the current tokens-per-second (TPS) output rate and consults a TPS-to-frequency table (profiled offline) to first narrow down to a coarse-grained frequency band. It

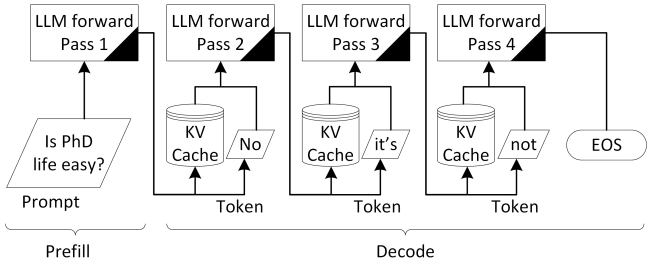


Figure 2. LLM Inference Serving: Prefill and Decode.

then applies fine-grained frequency adjustments in 15 MHz frequency steps (with hysteresis) to precisely maintain the 95th-percentile time-between-tokens (TBT) within the target bound, all while minimizing energy consumption. In essence, GreenLLM uses a fast TPS-feedback loop to throttle down the GPU when full speed is not needed during decoding, and to ramp up when needed to avoid violating latency SLOs (shown as Figure 1(b)GreenLLM, detail explained in Sect. 5.1.3).

By decoupling the two phases and tailoring the voltage-frequency settings of the GPU to the needs of each phase and each category of prompt length, GreenLLM substantially reduces node-level energy consumption for LLM inference. Our prototype implementation on A100 GPUs demonstrates up to 34% reduction in total energy consumption compared to NVIDIA's default DVFS policy on real LLM traffic traces (from Alibaba and Azure), with less than 3.5% queries missing their latency targets.

2 Background and Motivation

To optimize LLM serving, it is necessary to first understand its serving pipeline and how it interacts with modern GPU power management.

2.1 LLM Inference: Prefill vs. Decode

As illustrated in Figure 2, LLM inference comprises two distinct phases with different characteristics [50]. When a request arrives, the model first processes the entire input prompt in a prefill phase, then generates output tokens one-by-one in a decode phase. Prefill is compute-bound and $\sim O(n^2)$ in prompt length; it largely determines TTFT. This phase is typically latency-sensitive, since users await the model's first response token. In contrast, the decode phase unfolds as an iterative, stepwise generation: the model produces one token at a time, appending it to the context and repeating until completion. The runtime of the decode phase is proportional to the (unpredictable) length of the generated sequence, introduces a streaming aspect, the user gradually receives tokens, and the service often has a target time between tokens (TBT) to keep the output responsive.

These inherent differences mean that the optimal execution strategy can differ between prefill and decode. Prefill

benefits from aggressive resource usage to minimize one-time latency, whereas decode might afford more relaxed pacing as long as token stream timing stays smooth. Unfortunately, traditional power managers do not distinguish between these phases [12, 13, 26, 33, 34, 42, 43], treating an inference query as one monolithic task. This overlooks opportunities to specialize the scheduling and frequency of the GPU for the real needs of each phase.

2.2 GPU Frequency, Power, and Performance Trade-offs

Modern GPUs employ dynamic voltage and frequency scaling (DVFS) to balance performance and power consumption. GPU power consumption increases with both supply voltage and operating frequency, following the relation $P \propto V^2 f$ [9]. Since voltage scales approximately linearly with frequency, power can be approximated as $P \propto f^3$, providing substantial potential for energy reduction through frequency scaling.

To characterize these relationships for LLM inference, we profile Qwen3-14B on an A100-40GB DGX node using controlled microbenchmarks across varying frequencies and workload intensities (detailed in Section 4).

2.2.1 Profiling with microbenchmark. For a better profiling, we construct two trace-based microbenchmarks. Each replays short traces at fixed aggregate TPS targets, while we sweep the GPU **SM clock** via NVML app-clocks [22]; memory clocks are pinned and autoboost is disabled to isolate SM-frequency effects (details described in Sect. 4).

For **Prefill microbenchmark**, each trace runs prefill and then emits exactly one decoded token to terminate generation. Prompts are length-randomized within 256−1024 tokens. We set the TPS range from a 200 TPS to 30000 TPS, and for each TPS level, we vary the SM frequency from lowest 210MHz to highest 1410MHz.

For **Decode microbenchmark**, each trace begins with a very short prefill (32 tokens) and then decodes with per-stream generated lengths sampled from [256, 1024] tokens. We maintain the target TPS (200–3000) by adjusting concurrency and report the TBT versus the SM frequency. Traces are executed end-to-end (no pre-warmed KV cache).

Using these two microbenchmarks, we sweep SM clocks to quantify phase-specific frequency–power–latency trade-offs; Figures 3a–3c report the resulting U-shaped energy profiles and distinct knees for prefill and decode.

2.2.2 System Profiling. Figure 3a shows *normalized prefill energy* ($E/E_{\min}$) versus SM frequency for the prefill stage at different TPS levels. Most of the TTFT curves are convex: performance improves dramatically from low to mid-range frequencies, then flattens at higher frequencies. Energy drops sharply from very low clocks ($\leq$400 MHz) to ~0.9–1.0 GHz, reaches a broad minimum, and then rises again beyond ~1.1–1.2 GHz, yielding a clear U-shaped (convex) curve. This shape holds across loads (color-coded by TPS): higher TPS shifts the curves upward but the energy-minimizing frequency remains in a narrow band around 0.95–1.05 GHz, about 70–80% of the 1.41 GHz max (check detail explanation and math-metical analysis in Sect. 3.2).

Takeaway #1

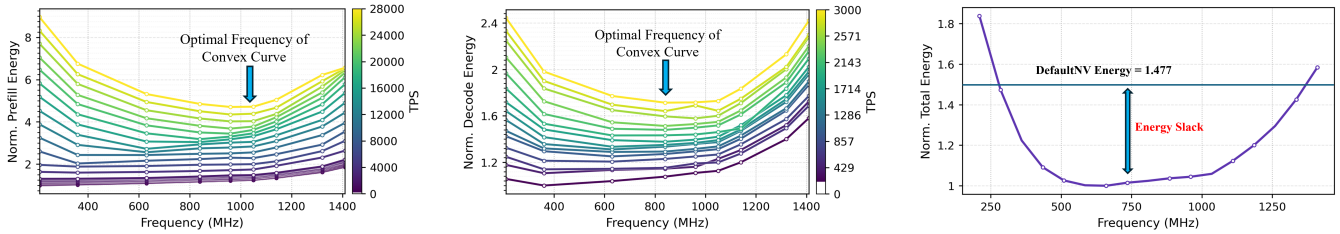
Prefill Energy shows a convex trend with a broad minimum around 0.95–1.05 GHz (≈70–80% of 1.41 GHz), indicating the headroom for frequency-energy optimization.

Figure 3b reports normalized decode energy ($E/E_{\min}$) versus SM frequency. The curves are convex: increasing frequency from low to moderate values reduces energy, but beyond ~1.2 GHz the energy rises again. Decode is largely *memory-bound* due to intensive key–value (KV) cache reads, so at high clocks GPU SMs spend more time stalled on the HBM/L2/NvLink fabric rather than on arithmetic units [13, 30, 48]. As a result, the time per token saturates with frequency, while the power grows superlinearly with the clock (roughly $P \propto fV^2$ [39]). The limited latency improvement coupled with higher power yields the observed U-shaped energy curve. Past the knee, decode energy increases more steeply than prefill, which remains more compute-bound and still gains meaningful speedups at higher clocks. Therefore, also, the optimal frequency point for decode worker shows a obvious lower value compared to the prefill worker.

Takeaway #2

The decode stage also shows a convex energy trend, suggesting room for optimization. However, its optimal frequency band is clearly lower than that of prefill, indicating that optimization should begin with a different strategy.

Furthermore, Figure 3c profiles total energy consumption from practical traces (chat traces with 5 queries per second from Alibaba, explained in Sect. 4) under different fixed frequency settings. The energy-frequency curve is also convex: running too slow (left side) prolongs execution (due to the SLO-violate latency) and inflates energy, while running at the highest frequency (right side) also increases energy due to disproportionate power draw, with an optimal sweet spot in between. In our measurements, an intermediate DVFS level minimized energy, for example, capping the GPU clocks around 0.75 GHz reduced total inference energy by ~47% compared to the default performance governor (which drives frequencies near peak). The lower optimal frequency compared to Fig.3a–3b stems from the trace's low decode TPS, which shifts the decode knee toward lower frequencies (e.g., Fig.3b at TPS ≈ 200).



(a) Normalized Prefill Energy vs. SM Frequency

(b) Normalized Decode Energy vs. SM Frequency

(c) Normalized Total Energy from Practical Trace vs. SM Frequency

Figure 3. System Energy/Frequency Profiling experiments with different traces

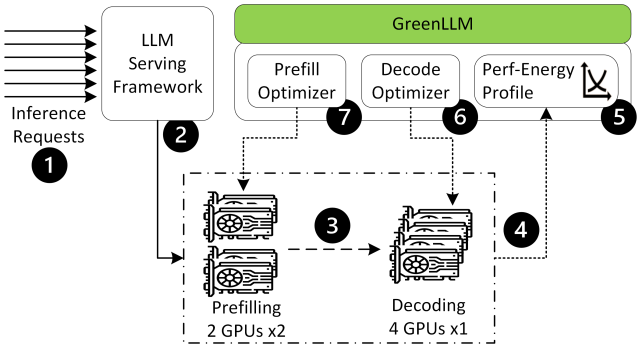


Figure 4. System Overview: Queue-aware prefill optimizer and dual-loop dynamic decode optimizer.

Takeaway #3

On practical traces, total energy is convex with a clear minimum, motivating SLO-aware power management.

These measurements reveal a consistent U-shaped relation between energy and frequency across prefill, decode, but with different knees for the two phases (Fig. 3a-b). Running prefill near a mid-frequency band minimizes energy while meeting TTFT, whereas decode should track the token stream and stay at a lower clock that just satisfies TBT. Consequently, any single governor is intrinsically suboptimal for LLM serving. This motivates GreenLLM: a dual-stage, SLO-aware optimization strategy that (i) isolates short and long prompts to protect TTFT, (ii) solves a queueing-aware optimization to pick prefill clocks per class, and (iii) uses a lightweight TPS/TBT feedback loop to hold decode latency while biasing clocks downward.

3 GreenLLM: SLO-Aware Dual-Stage LLM energy optimization

Figure 4 depicts GreenLLM's control planes. ❶ Inference requests arrive at the serving framework, which tokenizes and dispatches them to two execution pools. ❷ A Prefill pool (two workers, each using 2 GPUs) computes the KV-cache. ❸ Outputs are handed to a Decoding pool (four workers, 1 GPU each) for autoregressive generation. ❹ Per-worker telemetry, including throughput (TPS) and token-level latency SLOs—such as time-to-first-token (TTFT) and time-between-tokens (TBT)—is sampled and streamed to GreenLLM's ❺ Perf–Energy Profile block. The GreenLLM solves a queueing-based problem to select per-pool SM frequencies. GreenLLM issues DVFS updates to the decode ❻ and prefill ❼ pools, minimizing energy while meeting SLO constraints.

3.1 Adaptive Prompt Routing

Mixing short and long prompts in the same serving queue severely hurts latency due to head-of-line (HoL) blocking [29, 49]. When a lengthy prompt precedes shorter queries, those short requests get stuck waiting, resulting in a heavy long-tail latency distribution where many requests exceed their SLOs due to queueing delays rather than processing time.

To address this issue, we apply a simple length-based routing mechanism that partitions requests by prompt size and sends them to different specialized LLM workers. Instead of waiting for all queries in one queue, we configure $(n-1)$ threshold cut-off points on the prompt length to divide traffic among n workers based on the size of the input (n is 2 in our design). The intuition is straightforward: short prompts go to a worker optimized for speed on small inputs, while extremely long prompts go to a separate worker designed for heavy, long-sequence processing. To analytically explain the impact from mixed and separate prefill, we classify the prompts into two classes short-medium prompts (SM) and the long prompts (L). For example, with two prefill workers:

- **Short-Context Model:** Optimized for low-latency processing of SM inputs (up to *approximately 1024 tokens*). This worker can rapidly handle the common, shorter queries without being bogged down by extremely long tasks.
- **Long-Context Model:** Equipped to support much larger prompts and optimized for L-sequence processing. This worker handles the infrequent but expensive long queries on a separate track, so they don't block the short ones.

This simple length-based partitioning avoids the need for complex scheduling logic. By isolating long requests away from the main short-request queue, we eliminate most HoL blocking between dissimilar query lengths. The Orange histogram in Figure 5(a) shows the profiling experiment running

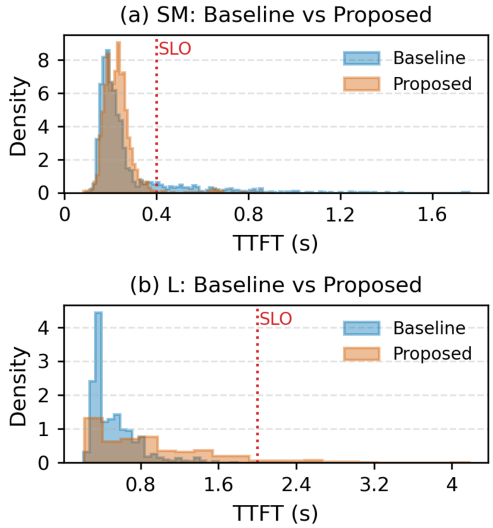


Figure 5. TTFT distribution before routing (a) and after length-based routing (b). Routing separates workloads by prompt length, lowering latency for short/medium queries while keeping long queries within their SLO.

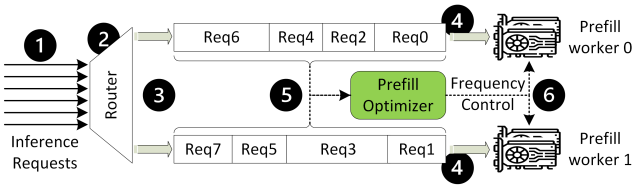


Figure 6. Prefill queue and control: prefill optimizer sets SM frequency to meet TTFT with lower energy

the Alibaba's chat trace with 8 queries-per-second on our system (check Sect. 4 for detail) with default method and the adaptive prompt routing. This length-aware routing dramatically reduces the SLO violations. The bulk of requests, which tend to have short or medium prompts, now achieve consistently low SLO violations because these short queries are no longer delayed by rare long requests in the same queue, so their tail latency drops significantly. Meanwhile, the long prompts are still processed in a reasonable time on their dedicated worker, but crucially their slower processing no longer interferes with the vast majority of traffic. By aligning each request to an appropriate execution path, we better satisfy divergent latency objectives across mixed workloads: fast service for short prompts and efficient handling of long prompts, with minimal mutual impact. In total, the percentage of the requests meeting the SLO increase sharply from 89.9% to 96.4%.

3.2 Prefill Modeling and Optimization

As mentioned in Figure 6 and Section 3.1, the queue is an important source of delay. We treat the observed queueing as direct information to start the optimization for the prefill stage. To act on this signal, we need a service-time model that predicts both latency and energy as a function of prompt length and GPU frequency.

To do that, we first profile the serving stack across a range of prompt lengths on a reference SM clock f_{ref} (typically the maximum clock). The prefill pass of a decoder-only Transformer has two dominant components per layer: (i) linear terms from QKV/output projections and the FFN, and (ii) a quadratic term from causal attention. The prefill FLOPs per layer can be summarized as

$$\boxed{\text{FLOPs}_{\text{prefill}/\ell}(B,n) \;=\; A\,n \;+\; C\,n^2,} \tag{1}$$

with

$$A = 8B\,d_{\text{model}}^2 + 4B\,d_{\text{model}}\,d_{\text{ff}},$$
$$C = 4\alpha B H_q d_k \;\approx\; 4\alpha B d_{\text{model}} \quad (\text{since } H_q d_k = d_{\text{model}}).$$

Interpretation: the linear term An comes from QKV/output projections and the FFN; the quadratic term Cn^2 comes from causal attention [38]. Here d_{model} is the hidden size, d_{ff} the FFN width, H_q the number of query heads, d_k the head dimension (typically $H_q d_k = d_{\text{model}}$), and $\alpha \approx \frac{1}{2}$ if the kernel computes only the causal triangle; otherwise $\alpha \approx 1$. Eq. (1) makes explicit that projections+FFN scale as $O(n)$ while attention scales as $O(n^2)$.

To model the prefill-stage latency, we simplify Eq. (1) into an interpretable quadratic in input length and profile the system at f_{ref} using a sweep of prompt lengths. Let L_k be the tokens of job k. We fit

$$t_k^{\text{ref}} \;\approx\; a\,L_k^2 + b\,L_k + c, \tag{2}$$

where a captures the attention cost, b captures projections+FFN, and c absorbs fixed overheads (e.g., tokenization and launches).

With real measurements of Qwen-14B, we therefore fit a second-order polynomial to the measured prefill latencies as a function of prompt length, as shown in Figure 7.

We then incorporate the effect of DVFS by modeling latency is inversely proportional to frequency (a reasonable first-order assumption since lower clock speeds linearly slow down compute-bound workloads). Thus, for a general frequency f, the prefill latency for job k is

$$t_k(f) \;=\; t_k^{\text{ref}} \cdot \frac{f_{\text{ref}}}{f}. \tag{3}$$

In practice, when t_k^{ref} is not directly measured for a new L_k, we use the fitted model in (2) as a surrogate. Using this per-job model, we can predict the total prefill *busy time* for a set of jobs as a function of frequency. Suppose that a batch of jobs (or all pending requests in a scheduling interval) is indexed by $k = 1, 2, \ldots, N$. The total busy time (cumulative time the GPU is actively running prefill computation for all these jobs) at frequency f is

$$\text{busy}(f) \;=\; \sum_{k=1}^{N} t_k(f). \tag{4}$$

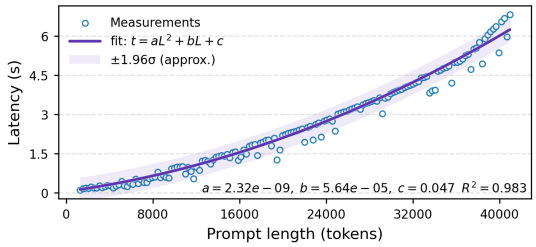


Figure 7. Prefill latency vs prompt length for Qwen3-14B; quadratic fit t = aL² + bL + c to measurements

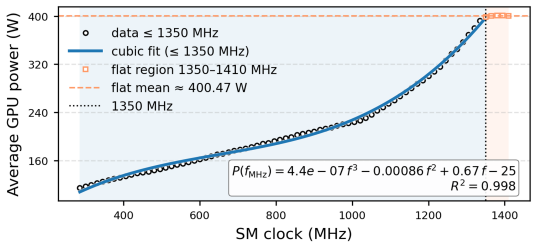


Figure 8. Power modeling with Qwen3-14B inference prefill with varying frequency; measured curve with cubic fit P(f) capturing DVFS scaling.

Substituting (3) gives

$$\text{busy}(f) \;=\; \sum_{k=1}^{N}\left(t_k^{\text{ref}}\cdot\frac{f_{\text{ref}}}{f}\right) \;=\; \frac{f_{\text{ref}}}{f}\sum_{k=1}^{N} t_k^{\text{ref}} \;=\; \frac{f_{\text{ref}}}{f}\,T_{\text{ref}}, \tag{5}$$

where $T_{\text{ref}} \triangleq \sum_{k=1}^{N} t_k^{\text{ref}}$ denotes the total prefill execution time for these jobs at the reference frequency.

In our SLO-aware setting, we have a latency target D (e.g., the maximum allowable time to finish all prefills, derived from the SLO). To meet the SLO, the chosen frequency must satisfy the requisite:

$$\text{busy}(f) \;\le\; D, \tag{6}$$

i.e., the total prefill work is completed by the deadline. Higher frequencies reduce busy time but consume more power, while lower frequencies save energy but increase runtime; our goal is to find the optimal balance with the following power model.

We model the power draw and energy consumption of the GPU as functions of frequency. From system profiling, the *active* power (while executing prefill) increases superlinearly with frequency, consistent with CMOS DVFS where dynamic power grows roughly cubically with frequency [39] (via joint voltage–frequency scaling). We thus fit a cubic polynomial to active power over frequency:

$$\begin{aligned} P(f) \;&=\; k_3 f^3 \;+\; k_2 f^2 \;+\; k_1 f \;+\; k_0, \\ P_{\text{idle}} \;&=\; P_0. \end{aligned} \tag{7}$$

where k_3, k_2, k_1, k_0 are regression coefficients. When the GPU is actively running prefill at frequency f, we use $P(f)$ as the instantaneous power draw. P_{idle} is the background power when the GPU is powered but idle. Note $P_0 \neq k_0$. To build the model, we drive the prefill tier with fixed-length prompts (e.g., 1,024 tokens) at a high request rate (40 QPS) to saturate the SMs, then sweep the SM clock and record GPU power. Figure 8 reports the measured power–frequency curve. We model average power with a low-order polynomial plus a frequency-independent baseline flat.

Consider an SLO interval of length D. Let $\text{busy}(f)$ denote the total prefill busy time (sum of per-job runtimes) at frequency f. The *active* energy is

$$E_{\text{active}}(f) \;=\; P(f)\,\text{busy}(f), \tag{8}$$

assuming the GPU draws $P(f)$ whenever it is busy. If $\text{busy}(f) \le D$, the remaining time $D - \text{busy}(f)$ is idle, yielding *idle* energy

$$E_{\text{idle}}(f) \;=\; P_{\text{idle}}\,[\,D - \text{busy}(f)\,], \qquad \text{when busy}(f) \le D, \tag{9}$$

otherwise the SLO is violated and such f is infeasible. The total energy within the SLO window is

$$E_{\text{total}}(f) \;=\; E_{\text{active}}(f) \;+\; E_{\text{idle}}(f). \tag{10}$$

Using the prefill busy-time model $\text{busy}(f) = \frac{f_{\text{ref}}}{f}\,T_{\text{ref}}$ with

$$T_{\text{ref}} \;\triangleq\; \sum_{k=1}^{N} t_k^{\text{ref}} \;\approx\; \sum_{k=1}^{N}\left(aL_k^2 + bL_k + c\right), \tag{11}$$

we obtain

$$\begin{aligned} E_{\text{total}}(f) &= (k_3 f^3 + k_2 f^2 + k_1 f + k_0)\,\frac{f_{\text{ref}}}{f}\,T_{\text{ref}} + P_{\text{idle}}\left[D - \frac{f_{\text{ref}}}{f}\,T_{\text{ref}}\right] \\ &= f_{\text{ref}}\,T_{\text{ref}}\left(k_3 f^2 + k_2 f + k_1 + \frac{k_0}{f}\right) + P_{\text{idle}}\left[D - \frac{f_{\text{ref}}}{f}\,T_{\text{ref}}\right]. \end{aligned} \tag{12}$$

Because $E_{\text{total}}(f)$ in (12) is non-monotonic, at runtime, GreenLLM selects the clock by solving a optimization problem under the constrain:

$$\min_{f\in[f_{\min},\,f_{\max}]} E_{\text{total}}(f) \quad \text{s.t. busy}(f) \le D.$$

At runtime, the *Queue Optimizer* solve the optimization problem dynamically and applies the best feasible frequencies on workers to achieve the optimal energy consumption under the SLO constrains.

3.3 Decode-Stage optimization

To minimize energy consumption during autoregressive decoding without degrading user-visible performance, we design a decode stage controller that operates in two coordinated loops, as shown in Figure 9. We will introduce these two loops in the following subsections. Note that all decisions are made outside the GPU execution path by an asynchronous process that stays outside the critical path.

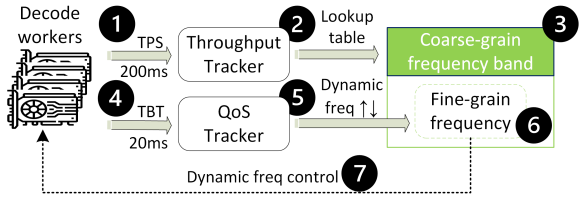


Figure 9. Decode control: TPS determines coarse frequency-band and fine frequency adjustment with hysteresis to meet P95 TBT.

3.3.1 Coarse-Grain Frequency Band Selection.

The goal of the coarse-grain loop is to estimate decoding load and swiftly select a suitable GPU frequency range that maintains TBT targets with minimal energy consumption.

We track TPS using a sliding window of the past 200 ms of emitted tokens. This window smooths out momentary bursts and provides a stable estimate of overall decode throughput. To determine optimal frequencies, we conduct an offline profiling sweep using microbenchmark with varying TPS value from 200 to 3000TPs (Sect. 2.2.1 and Figure 3b) across a range of GPU SM clocks. For each TPS bucket, we identify the lowest frequency that satisfies two criteria, namely, maintains P95 TBT below 100 ms and minimizes energy per token (i.e., power/TPS). These values form a static lookup table: *TPS bucket → optimal frequency*. At runtime, the controller:

1. Map the current TPS to its corresponding bucket, as shown in Figure 9 ❶, ❷.
2. Select the frequency band as the triplet: the optimal frequency plus its two neighbors (e.g., $[f_{\text{lo}}, f_{\text{mid}}, f_{\text{hi}}]$)(❸).
3. Apply hysteresis: the band is updated only if the TPS remains in the new bucket for at least three consecutive 200 ms intervals. This mechanism balances reactivity with stability, preventing unnecessary frequency jitter.

3.3.2 Fine-Grain TBT Tracker and Frequency Control.

The fine-grain loop makes fine adjustments to GPU frequency every 20 ms based on real-time token latency observations(Figure 9 ❹). Its role is to track the target TBT (100 ms P95) and conserve energy by reducing the frequency whenever possible. For every 20 ms, we compute the P95 TBT in a sliding window of recent tokens. Let

$$\text{TBT}_{\text{margin}} = \frac{\text{P95 TBT}}{T_{\text{SLO}}}, \qquad T_{\text{SLO}} = 100\,\text{ms}. \tag{13}$$

The update rule is(Figure 9(❺, ❻):

- If $\text{TBT}_{\text{margin}} > 1.0$: increase GPU frequency by 15 MHz (up to the band upper bound).
- If $\text{TBT}_{\text{margin}} < 0.65$: decrease frequency by 15 MHz (not below the band lower bound).
- Else: hold the frequency.

Table 1. Hardware and software configuration.

Component	Setting
Node	NVIDIA DGX–A100 (1 node) [19]
GPUs	8× A100–SXM4 40 GB; intra-node NVLink/NVSwitch
CPU	AMD EPYC 7302, 16 cores @ 3.0 GHz [2]
Framework	Dynamo v0.3.1 [20]
Inference kernel	NVIDIA TensorRT / TensorRT-LLM [23]
DVFS control	NVML application clocks (SM); memory clocks pinned [22, 24]

Each fine-grain adjustment is rate-limited to 15-30 MHz per control tick to preserve stability. The set point is constrained to the frequency band selected by the coarse-grain loop (Figure 9 ❸), based on the current TPS, and its two neighboring bands. The coarse- and fine-grain loops operate in concert: the coarse loop chooses the band, and the fine loop determines the optimal frequency within it.

3.3.3 TBT Tracker and Coarse-Grain Frequency Band Update.

To address performance drift and workload variability, we update the coarse lookup table using feedback from the fine-grain controller. Every 6 seconds, we sample adjusted frequency events and compute TBT across the window. We apply adaptation logic to shift frequency bands upward or downward when sustained bias is detected (>80% of adjustments exceed band bounds). This dual-loop mechanism enables fine-grained, SLO-aware control that responds to decoding dynamics in real time.

4 Experimental Setup

4.1 Hardware and Software Setup

As shown in Table 1, all experiments run on a NVIDIA DGX-A100 node, using NVIDIA Dynamo as deployment framework.

4.2 Models and Inference Traces

4.2.1 Models.

We evaluate using a dense model (Qwen3-14B) and a mixture-of-experts model (Qwen3-30B-MoE) [31, 32, 36], shown in Table 2. The system uses two execution pools: a Prefill pool (2 workers, 2 GPUs each) and a Decode pool (4 workers, 1 GPU each).

4.2.2 Traces and Evaluation Methods.

We evaluate using: (i) phase-specific microbenchmarks that isolate prefill and decode performance across frequency ranges, and (ii) production traces from Alibaba ServeGen [44] (QPS: 1,3,5,8,10) and Azure 2024 [17] (downsampled to 1/8, 1/5, 1/4 rates of

Table 2. Models: Qwen3-14B and Qwen3-30B, datatype: BF16.

Model	Type	Params (total / active)	Layers	Experts (active)
Qwen3−14B	Dense	14.8B / 14.8B	40	—
Qwen3−30B	MoE	30.5B / 3.3B	48	128 (8)

Note: "Active" parameters are used per token (for dense models, active = total).

its original rate to match single-node capacity (the original targets a GPU cluster) while preserving the inter-arrival structure).

We compare three configurations: **DefaultNV** (NVIDIA default), **PrefillSplit** (length-based routing only), and **GreenLLM** (length-based routing + prefill/decode optimizer). SLOs target TTFT < 400ms for Short/Medium prompts, < 2s for Long, and P95 TBT ≤ 100ms during decode by following Azure targets [34].

5 Results

We evaluated GreenLLM's energy–performance tradeoffs through a set of targeted experiments. Our evaluation follows three steps:

(1) Phase-level profiling: We use microbenchmarks to measure the two main inference stages separately: prompt processing (prefill) and token generation (decode). This step clearly demonstrates the different principles, strategies, and benefits for different phases of LLM serving.

(2) End-to-end validation: We then run complete workloads to check whether GreenLLM can meet service-level objectives (SLOs) under realistic conditions at the system level.

(3) Margin analysis: Furthermore, we study how latency slack affects the balance between energy use and SLO compliance, showing the adaptability of our design across a wide range of scenarios.

5.1 Phase-level profiling

5.1.1 Prefill-Stage Evaluation. We first evaluate GreenLLM's prefill optimization under varying prompt lengths and input rates. Figure 10a–c present the average time-to-first-token (TTFT) for the Short, Medium, and Long prompt classes across increasing synthetic load (TPS), comparing GreenLLM's adaptive frequency tuning method against NVIDIA's default DVFS governor (**"defaultNV"**). We sweep throughput (TPS) to simulate different load levels, measuring P90 TTFT and energy at each point using the microbenchmark as Sect. 4. As shown by the line markers in Fig. 10a–b, for short/medium prompts (400 ms SLO), GreenDVFS intentionally save energy by exploiting the slack from SLO: TTFT is slightly higher than the default at low–mid TPS, and the energy-saving curve (green) starts ~10–20% and gradually

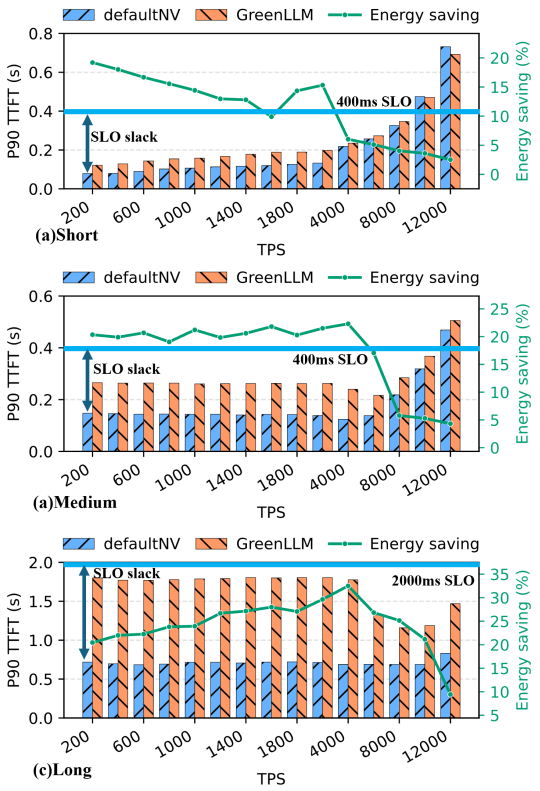


Figure 10. Prefill microbenchmarks (TTFT vs TPS) with defaultNV and GreenLLM. (a) Short; (b) Medium; (c) Long.

falls to ~0%, then collapses near saturation when the controller returns to high clocks to protect the SLO. For long prompts (bottom; 2s SLO), the larger prefill cost creates more usable slack: energy savings rise toward ~25–30% at mid load, and GreenLLM increases the TTFT much higher to save the energy consumption; as TPS climbs further, the slack shrinks. Across all three, TTFT stays within the class SLO through most of the range and energy savings diminish when the system nears saturation. The microbenchmarks reveal that the defaultNV often achieves TTFT well below the SLO, leaving substantial SLO slack. This slack represents performance headroom that can be traded for energy savings. For instance, on an A100 GPU a moderate-sized request might yield a first-token latency of only ~75ms against a 400 ms SLO, implying 325 ms of slack for possible energy saving.

5.1.2 Decode-Stage Evaluation. Next, we examine the decoding performance under stable token generation demand, to evaluate GreenLLM's runtime controller. In this experiment, a fixed number of concurrent decode streams yields a constant aggregate token rate (TPS), which we vary from 200 up to 3000 tokens/s. GreenLLM's token-tracking DVFS controller is enabled to dynamically adjust the SM frequency every 20 ms, and we compare its results to the **"defaultNV"**.

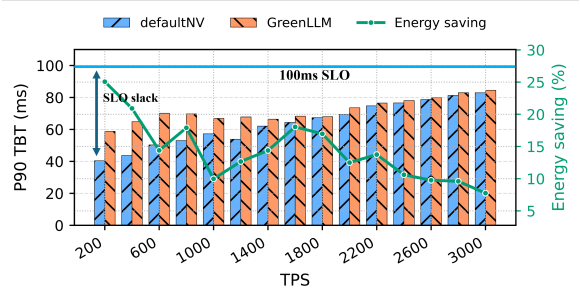


Figure 11. Decode microbenchmarks (TBT vs TPS) with defaultNV and GreenLLM.

Figure 11 reports P90 time-between-tokens (TBT) across a decode TPS sweep together with GreenLLM's GPU energy reduction. Across 200–3000 TPS, GreenLLM's P90 TBT closely tracks the defaultNV and remains within the 100 ms SLO at all points. The gap is most visible only at very light load: at 200–400 TPS GreenLLM shows ~60–66 ms versus ~40–46 ms for the defaultNV, still comfortably below the SLO. As load increases, the difference shrinks—around 1000 TPS we observe ~65 ms (GreenLLM) vs. ~58 ms (defaultNV)—and from 1800 TPS onward the bars are nearly indistinguishable; at 3000 TPS both converge near ~85–86 ms. The energy savings are highest at low TPS (~20–25%) and decrease with load, reaching ~8–12% at 2400–3000 TPS, with a modest rebound around 1600–1800 TPS due to the controller's band selection. In short, the decode-phase controller maintains SLO-compliant token latency while cutting GPU energy by 8–25%, with the greatest gains when the workload offers more headroom.

Takeaway #4

GreenLLM reduces power while staying close to the SLO target by using a lower frequency. Its scheduler ensures TTFT always meets, but does not exceed, the SLO.

5.1.3 Dynamic-Tracing Evaluation. Furthermore, we assess GreenLLM's ability under a time-varying workload to verify its adaptability and stability. Figure 1 (in Sect. 1) drives decoding with a synthetic sinusoidal TPS target to test tracking and stability. The top panel shows **"defaultNV"** (NVIDIA's scaling governor): the SM clock sits almost stationary in a narrow high band (~1.1-1.4 GHz) and does not follow changing TPS, confirming the lack of TPS-aware adaptation. In contrast, the bottom figure shows GreenLLM's DVFS decisions: the SM clock tracks the workload increase from ~450 MHz to ~1.35 GHz as TPS increases, then symmetrically reverses as TPS falls with small rate-limited adjustments from the 20 ms fine loop and 15 MHz steps. This is consistent with our controller design (Sect. 3.3): we measure TPS continuously and pick a coarse frequency band, then apply fine, hysteretic nudges to hold TBT within its SLO while minimizing frequency, and thus energy, whenever slack is available. Across the run, p99 TBT stays ≤ 100 ms under both policies (GreenLLM 83.2 ms vs. defaultNV 84.6 ms), demonstrating SLO-compliant tracking with lower clocks when demand allows. In the same experiment, GreenLLM achieves 8.9% lower decode energy while achieving similar or slightly better tail latency (p99 TBT 83.2 ms vs. 84.6 ms).

Takeaway #5

GreenLLM's dual-loop decode controller design can successfully traces workload intensity and SLO dynamically, modulating SM clocks in real time to save energy.

5.2 Trace Evaluation with Intensity Scaling

We assess GreenLLM on two production traces, **Alibaba** and **Azure**, under multiple intensity levels to capture realistic burstiness and prompt-length skew. We replay the Alibaba trace at {1, 3, 5, 8, 10} QPS for at least 30 minutes per run, and We downsample the May 2024 Azure trace to {1/8, 1/5} of its original rate to match single-node capacity (the original targets a GPU cluster) while preserving the inter-arrival structure. SLOs target **TTFT** < 400 ms for S/M and < 2 s for L, and **TBT** < 100 ms during decoding [34]. We collect the pass rates (TTFT%, TBT%), together with energy for prefill and decode. We compare three configurations: DefaultNV, PrefillSplit, and GreenLLM introduced in Sect. 4.2. All experiments are run with two representative LLMs: a dense model Qwen3-14B and a mixture-of-experts model, Qwen3-30B-MoE, introduced in Table 2.

Table 3 shows results with Qwen3-14B. Energy savings of GreenLLM drop from 27.5% at 1 QPS to 6.8% at 10 QPS as higher decode clocks and power are needed to sustain TBT. On Azure, however, GreenLLM consistently saves 28–34%, mainly from reduced decode energy (0.62–0.73× defaultNV). Similar trends hold for Qwen3-30B-MoE in Table 4: total energy falls by 19.7–31% on Azure and 10–21% on Alibaba, with decode energy at 0.73–0.89× defaultNV.

Across traces and intensities, **GreenLLM** maintains high SLO pass rates. For Alibaba chat, Table 3 shows that TTFT%, TBT% remain ≥95% through 1–8 QPS. Especially for QPS 8, the TTFT% from **defaultNV** shows much worse result, where it is only 89.9% compared to 95.7% in GreenLLM. At 10 QPS, GreenLLM's pass rates drop to TTFT 88.2% and TBT 90.9%—consistent with the system nearing prefill saturation, at which point the decode optimizer raises clocks to protect streaming quality.

On Azure code (Qwen3-14B), rest part of Table 3, **GreenLLM** retains TBT 100% while TTFT% varies with long-prompt mix (e.g., 94.2% on `code5`); conversation workloads maintain ~99–100% for both TTFT and TBT. For Qwen3-30B-MoE, demonstrated in Table 4, pass rates are similarly high at feasible intensities with occasional modest TBT% reductions (e.g., 93.8–97.0%) on Azure code slices.

Table 3. Energy and SLOs on chat workloads from Qwen-14B (energies normalized to defaultNV

Workload	Method	Rel. Decode	Rel. Prefill	TTFT (%)	TBT (%)	ΔEn (%)
	defaultNV	1.000	0.589	99.3	98.2	0.00
chat_1qps	PrefillSplit	0.998	0.586	98.5	100.0	0.28
	GreenLLM	0.675	0.479	98.9	99.6	27.52
	defaultNV	1.000	0.903	99.5	98.6	0.00
chat_3qps	PrefillSplit	0.995	0.878	98.8	98.6	1.61
	GreenLLM	0.798	0.714	99.2	95.1	20.53
	defaultNV	1.000	1.072	96.0	98.6	0.00
chat_5qps	PrefillSplit	1.006	1.046	99.0	98.6	0.94
	GreenLLM	0.832	0.865	98.2	96.0	18.13
	defaultNV	1.000	1.314	89.9	96.1	0.00
chat_8qps	PrefillSplit	1.023	1.292	96.4	96.4	−0.04
	GreenLLM	0.974	1.083	95.7	95.1	11.16
	defaultNV	1.000	1.213	84.9	90.7	0.00
chat_10qps	PrefillSplit	1.008	1.183	89.1	91.3	1.03
	GreenLLM	1.029	1.034	88.2	90.9	6.78
	defaultNV	1.000	1.707	98.5	100.0	0.00
Azure_code5	PrefillSplit	0.964	1.663	96.6	100.0	2.95
	GreenLLM	0.629	1.294	94.2	100.0	28.98
	defaultNV	1.000	1.696	99.9	100.0	0.00
Azure_code8	PrefillSplit	0.981	1.652	100.0	100.0	2.35
	GreenLLM	0.728	1.129	99.9	100.0	31.12
	defaultNV	1.000	1.393	99.7	100.0	0.00
Azure_conv5	PrefillSplit	0.992	1.370	100.0	100.0	1.32
	GreenLLM	0.624	0.954	99.8	100.0	34.09
	defaultNV	1.000	1.416	100.0	100.0	0.00
Azure_conv8	PrefillSplit	0.993	1.373	100.0	100.0	2.07
	GreenLLM	0.713	0.922	100.0	100.0	32.31

Table 4. Energy and SLOs on chat workloads from Qwen-30B (MoE) (energies normalized to defaultNV

Workload	Method	Rel. Decode	Rel. Prefill	TTFT (%)	TBT (%)	ΔEn (%)
	defaultNV	1.000	0.650	100.0	100.0	0.00
chat_1qps	PrefillSplit	1.010	0.643	100.0	100.0	−0.16
	GreenLLM	0.731	0.580	100.0	99.6	20.56
	defaultNV	1.000	0.790	99.4	97.3	0.00
chat_3qps	PrefillSplit	1.001	0.784	99.2	97.3	0.30
	GreenLLM	0.855	0.712	99.2	95.6	12.46
	defaultNV	1.000	0.872	99.3	97.9	0.00
chat_5qps	PrefillSplit	1.004	0.834	99.1	98.0	1.82
	GreenLLM	0.890	0.791	99.3	96.2	10.23
	defaultNV	1.000	1.063	99.9	99.6	0.00
Azure_conv5	PrefillSplit	0.995	1.051	99.7	99.7	0.82
	GreenLLM	0.808	0.849	99.7	96.1	19.67
	defaultNV	1.000	0.921	99.9	99.8	0.00
Azure_conv8	PrefillSplit	0.997	0.908	100.0	99.7	0.88
	GreenLLM	0.783	0.706	100.0	93.8	22.51
	defaultNV	1.000	1.140	100.0	99.7	0.00
Azure_code5	PrefillSplit	0.993	1.112	99.9	99.8	1.57
	GreenLLM	0.642	0.883	100.0	95.9	28.76
	defaultNV	1.000	0.985	99.6	99.8	0.00
Azure_code8	PrefillSplit	1.001	0.952	100.0	100.0	1.61
	GreenLLM	0.655	0.713	99.6	97.0	31.05

The routing-only ablation **PrefillSplit** reduces head-of-line interference and slightly tightens TTFT tails, but yields only $\lesssim$1–3% energy change across traces and models. In contrast, **GreenLLM**'s *phase-aware DVFS* (prefill optimizer + decode optimizer) delivers the decisive savings: on Azure with Qwen3-14B, decode energy falls to 0.63–0.71× Default; with Qwen3-30B-MoE it falls to 0.64–0.80×, all while sustaining high TTFT/TBT pass rates (Tables 3–4). These results validate that exploiting the prefill/decode split, and controlling frequency using dedicated strategy for prefill/decode, is the right lever for energy proportionality under SLOs.

Takeaway #6

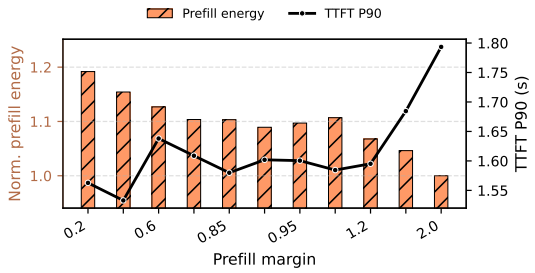
Across all model families and traces, GreenLLM consistently reduces GPU energy at scale without sacrificing throughput and with high SLO compliance.

5.3 SLO Margin Sensitivity: Energy–SLO Tradeoffs

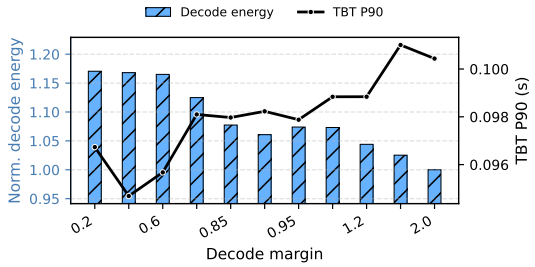
We next evaluate how the latency SLO margin, the slack or tightness around target latencies, affects GreenLLM's energy–latency tradeoffs. In this experiment, we keep these SLO targets by scaling them with margin factors (as described at Sect. 3, for prefill, the margin factor is coefficient applied on latency deadline D) ranging from an extremely strict 0.2× up to a highly relaxed 2.0× (with intermediate values 0.6×, 0.85×, 0.95×, and 1.2×), and measure the impact on GPU energy consumption and tail latency (90th percentile). Here we use the Alibaba chat trace (10 QPS) on Qwen-14B model.

We first adjust the TTFT margin factor while holding the decode-phase margin at a representative margin of 0.95× (nearly the baseline target). Figure 12a illustrates the total energy consumed in the prefill stage (lines) and the achieved P90 TTFT (bars) for each margin setting. As expected, looser margin reduce energy usage and increase the TTFT. At the strict end (e.g., margin 0.6×, a 40% tighter deadline), GreenLLM's prefill controller must raise clock frequencies to speed up computation processing, showing a much lower P90 TTFT, however, result in higher prefill energy consumption. Conversely, looser prefill margin yield substantial energy savings. At a relaxed margin like 1.2× (20% more slack), or even at 2.0×, which doubles the latency allowance, the prefill controller can dial down GPU utilization while almost violate latency requirement. Under these high-margin settings, the P90 TTFT increases, for example, rising from less than ∼1000 ms at baseline to almost ∼1640 ms at 1.2×, staying well within the allowed SLO buffer.

Furthermore, we perform a similar experiment for the decode stage, varying the TBT margin while fixing the prefill margin at 0.95×. Figure 12b plots the total decode energy (bars) and P90 TBT (lines) as we sweep the decode margin from tight (0.2×) to relaxed (2.0×). The trends here closely mirror the prefill-phase behavior. Tightening the decode budget forces GreenLLM to increase GPU frequency during token generation, which drives up energy usage. For example, at a strict 0.85× decode margin, the controller ramps up to maintain a faster TPS, and P90 TBT improves to meet the ∼85–90 ms target with no SLO violations. In contrast, a relaxed decode SLO (e.g., 1.2× or 2.0×) saves more energy at lower clocks. The 90th-percentile token latency grows only slightly under these looser settings (reaching ∼110–120 ms



(a) Prefill margin with fixed Decode margin $x0.95$.



(b) Decode margin with fixed prefill margin $x0.95$.

Figure 12. Margin sensitivity: energy–latency tradeoffs in prefill and decode.

at 1.2×), and violates the acceptable SLO for streaming outputs. These margin-sensitivity experiments demonstrate that GreenLLM automatically tunes its behavior to both stricter and more lenient latency requirements, enabling a smooth energy–latency tradeoff without any manual reconfiguration.

Takeaway #7

GreenLLM preserves SLO compliance across diverse latency budgets, achieving SLO-aware energy optimization automatically without re-engineering.

6 Related Work

6.1 LLM Disaggregation and Scheduling

LLM inference consists of a compute-intensive prefill phase and a sequential, memory-bound decode phase, whose differing resource profiles complicate efficient serving [3, 14, 25, 41, 50]. Splitwise [27] addresses this by disaggregating the two phases across distinct "prompt" and "token" machines, transferring KV caches over high-speed links to exploit heterogeneous clusters and improve throughput and cost efficiency. Other scheduling techniques instead focus on the decode phase: Orca [45] introduced in-flight batching to insert new requests into ongoing decode iterations, improving utilization and latency. More recently, Jaillet et al. modeled decode scheduling under GPU memory constraints and proposed online algorithms that reduce latency and energy by better managing prompt lengths and KV cache usage [10]. These efforts primarily optimize batching and machine allocation for throughput or latency. In contrast, GreenLLM manages phases within a single GPU, combining prompt-length–aware queueing and phase-specific DVFS rather than relying on static phase placement or batching alone.

6.2 GPU Power Management

DVFS is a central mechanism for trading performance and energy, and has been studied extensively for ML workloads. Nabavinejad et al. [18] coordinated batching delays with GPU DVFS to save energy in CNN inference, although such coarse-grained control is poorly suited to unpredictable LLM decode loops. Wang et al. [42] showed that Huawei's Ascend NPUs support millisecond-level frequency control, enabling per-operator DVFS guided by accurate power–performance models; this fine-grained tuning reduced core power by over 13% with negligible slowdown. Patel et al. [26] characterized the usage of LLM inference power and built POLCA, which oversubscribes GPU power budgets at cluster scale to host more LLM servers under datacenter limits. Wilkins et al. [43] developed workload-level energy models parameterized by prompt and output lengths, and used them to design an offline energy-optimal heterogeneous scheduler. Together, these works demonstrate the value of analytical models and adaptive DVFS at both the operator and cluster levels. GreenLLM adopts a complementary approach: builds compact latency-power models online for the prefill phase, and applies queueing-aware optimization to set GPU frequencies dynamically, extending model-driven DVFS to runtime per-phase control on commodity GPUs.

6.3 SLO-Aware Energy Optimization

The scale of LLM deployment has made inference the dominant contributor to lifecycle energy, motivating techniques that reduce consumption while meeting latency SLOs. Kakolyris et al. [13] proposed an iteration-level DVFS controller that tracks the decode loop and adjusts the GPU frequency to maintain 99th percentile latency targets, producing up to 45% energy savings. An extended version, throttLL'eM [12], further incorporated KV-cache projections and autoscaling to cut cluster energy by over 40% under latency bounds. Complementary work has quantified the environmental footprint of inference, showing that large models can consume tens of watt-hours per query, emphasizing the urgency of runtime optimizations [12, 13]. Beyond DVFS, model- or cluster-level schedulers such as ClockWork [8], and DynamoLLM [34] coordinate batching, autoscaling, and frequency scaling to improve cost and performance efficiency, though they do not exploit LLMs' two-phase structure. Compared to these efforts, GreenLLM integrates SLO awareness directly into phase-specific control: prompt-length–based queuing reduces time-to-first-token, latency–power modeling informs prefill DVFS, and a token-tracking hysteresis controller adapts decode

frequency. This unified design enables substantial energy savings while respecting stringent service-level objectives.

7 Conclusion

In the paper, we have proposed **GreenLLM**, an adaptive SLO-aware dynamic frequency scaling framework for energy-efficient LLM serving. Our framework couples 1. length-aware routing to mitigate head-of-line effects and preserve TTFT for short prompts, 2. a queueing-aware, model-driven optimizer that selects energy-minimal SM clocks for prefill within latency constraints, and 3. a lightweight, dual-loop token-tracking DVFS controller that holds decode TBT targets while minimizing energy. The approach runs on commodity GPUs, requires no model changes, and integrates cleanly with existing serving stacks.

Across Qwen-14B (Desne) and Qwen-30B (MoE) on real Alibaba/Azure traces, GreenLLM consistently reduces node energy while maintaining throughput and high SLO pass rates, with the largest gains when decode has SLO slack room. The total energy drops by **10–34%** compared with Nvidia's default governor. Looking ahead, GreenLLM's principles can extend to larger clusters and next-generation GPUs for better efficiency and sustainability.

References

[1] 174 Power Global. 2025. How AI Changes Data Center Design Forever. https://174powerglobal.com/blog/how-ai-changes-data-center-design-forever/. Accessed: 2025-08-18.

[2] Advanced Micro Devices, Inc. 2019. *AMD EPYC™ 7002 Series—Data Sheet.* Technical Report. AMD. https://www.amd.com/content/dam/amd/en/documents/products/epyc/amd-epyc-7002-series-datasheet.pdf

[3] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 117–134.

[4] Reza Yazdani Aminabadi, Samyam Rajbhandari, Minjia Zhang, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Jeff Rasley, Shaden Smith, Olatunji Ruwase, and Yuxiong He. 2022. DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *SC '22: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Dallas, TX, USA. doi:10.1109/SC41404.2022.00051

[5] Antje Barth. 2023. Amazon EC2 Inf2 Instances for Low-Cost, High-Performance Generative AI Inference are Now Generally Available. https://aws.amazon.com/blogs/aws/amazon-ec2-inf2-instances-for-low-cost-high-performance-generative-ai-inference-are-now-generally-available/. AWS News Blog.

[6] Rani Borkar, Andrew Wall, Prasanth Pulavarthi, and Yuan Yu. 2024. Azure Maia for the era of AI: From silicon to software to systems. https://azure.microsoft.com/en-us/blog/azure-maia-for-the-era-of-ai-from-silicon-to-software-to-systems/. Microsoft Azure Blog.

[7] Joel Coburn, Chunqiang Tang, Adam Hutchin, Ajit Mathews, Alex Mastro, Amin Firoozshahian, et al. 2025. Meta's Second Generation AI Chip: Model-Chip Co-Design and Productionization Experiences. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA '25)*. ACM, Tokyo, Japan. doi:10.1145/3695053.3731409

[8] Arpan Gujarati, Reza Karimi, Safya Alzayat, Wei Hao, Antoine Kaufmann, Ymir Vigfusson, and Jonathan Mace. 2020. Serving DNNs like clockwork: Performance predictability from the bottom up. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 443–462.

[9] Sarah Harris and David Harris. 2021. *Digital Design and Computer Architecture, RISC-V Edition.* Morgan Kaufmann.

[10] Patrick Jaillet, Jiashuo Jiang, Konstantina Mellou, Marco Molinaro, Chara Podimata, and Zijie Zhou. 2025. Online Scheduling for LLM Inference with KV Cache Constraints. *arXiv preprint arXiv:2502.07115* (2025).

[11] Nidhal Jegham, Marwen Abdelatti, Lassad Elmoubarki, and Abdeltawab Hendawi. 2025. How Hungry is AI? Benchmarking Energy, Water, and Carbon Footprint of LLM Inference. *arXiv preprint arXiv:2505.09598* (2025).

[12] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Petros Vavaroutsos, Sotirios Xydis, and Dimitrios Soudris. 2025. throttLL'eM: Predictive GPU Throttling for Energy Efficient LLM Inference Serving. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1363–1378.

[13] Andreas Kosmas Kakolyris, Dimosthenis Masouros, Sotirios Xydis, and Dimitrios Soudris. 2024. SLO-Aware GPU DVFS for Energy-Efficient LLM Inference Serving. *IEEE Comput. Archit. Lett.* 23, 2 (July 2024), 150–153. doi:10.1109/LCA.2024.3406038

[14] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP '23)*. ACM, Koblenz, Germany, 611–626. doi:10.1145/3600006.3613165

[15] McKinsey & Company. 2023. The Economic Potential of Generative AI: The Next Productivity Frontier. https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/the-economic-potential-of-generative-ai-the-next-productivity-frontier. Accessed: 2025-08-18.

[16] Microsoft. 2023. General availability of Azure OpenAI Service expands access to large AI models with added enterprise benefits. https://azure.microsoft.com/en-us/blog/general-availability-of-azure-openai-service-expands-access-to-large-advanced-ai-models-with-added-enterprise-benefits/. States that ChatGPT runs inference on Azure AI infrastructure.

[17] Microsoft Azure. 2024. Azure LLM Inference Dataset 2024. https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2024.md. Microsoft Azure Traces; week-long sample collected May 2024.

[18] Seyed Morteza Nabavinejad, Sherief Reda, and Masoumeh Ebrahimi. 2022. Coordinated batching and DVFS for DNN inference on GPU accelerators. *IEEE transactions on parallel and distributed systems* 33, 10 (2022), 2496–2508.

[19] NVIDIA. 2020. *NVIDIA A100 Tensor Core GPU Architecture.* Technical Report. NVIDIA. https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf Ampere architecture; A100 memory/form factors (incl. SXM4 40GB).

[20] NVIDIA. 2025. NVIDIA Dynamo: Distributed Inference Framework. https://developer.nvidia.com/dynamo Project overview; this work uses v0.3.1.

[21] NVIDIA. 2025. NVIDIA H100 Power Consumption Guide. https://resources.nvidia.com/en-us-gpu-resources/h100-datasheet-24306. Accessed: 2025-08-18.

[22] NVIDIA. 2025. *NVIDIA Management Library (NVML).* https://developer.nvidia.com/management-library-nvml Accessed: August 2025.

[23] NVIDIA. 2025. *NVIDIA TensorRT Documentation*. https://docs.nvidia.com/deeplearning/tensorrt/latest/architecture/architecture-overview.html

[24] NVIDIA. 2025. `nvidia-smi` *User Guide*. https://docs.nvidia.com/deploy/nvidia-smi/index.html Application clocks control via `nvidia-smi -applications-clocks`.

[25] NVIDIA Developer. 2023. Mastering LLM Techniques: Inference Optimization. https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization/.

[26] Pratyush Patel, Esha Choukse, Chaojie Zhang, Íñigo Goiri, Brijesh Warrier, Nithish Mahalingam, 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*. 207–222.

[27] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 118–132.

[28] Pratyush Patel, Zibo Gong, Syeda Rizvi, Esha Choukse, Pulkit Misra, Thomas Anderson, and Akshitha Sriraman. 2023. Towards improved power management in cloud gpus. *IEEE Computer Architecture Letters* 22, 2 (2023), 141–144.

[29] Archit Patke, Dhemath Reddy, Saurabh Jha, Haoran Qiu, Christian Pinto, Chandra Narayanaswami, Zbigniew Kalbarczyk, and Ravishankar Iyer. 2024. Queue management for slo-oriented large language model serving. In *Proceedings of the 2024 ACM Symposium on Cloud Computing*. 18–35.

[30] Ramya Prabhu, Ajay Nayak, Jayashree Mohan, Ramachandran Ramjee, and Ashish Panwar. 2025. vattention: Dynamic memory management for serving llms without pagedattention. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 1133–1150.

[31] Qwen Team. 2025. Qwen/Qwen3-14B. https://huggingface.co/Qwen/Qwen3-14B Model card.

[32] Qwen Team. 2025. Qwen/Qwen3-30B-A3B-Instruct-2507. https://huggingface.co/Qwen/Qwen3-30B-A3B-Instruct-2507 Model card; A3B MoE variant.

[33] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Esha Choukse, Haoran Qiu, Rodrigo Fonseca, Josep Torrellas, and Ricardo Bianchini. 2025. Tapas: Thermal-and power-aware scheduling for LLM inference in cloud platforms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1266–1281.

[34] Jovan Stojkovic, Chaojie Zhang, Íñigo Goiri, Josep Torrellas, and Esha Choukse. 2025. Dynamollm: Designing llm inference clusters for performance and energy efficiency. In *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 1348–1362.

[35] Zhenheng Tang, Yuxin Wang, Qiang Wang, and Xiaowen Chu. 2019. The impact of GPU DVFS on the energy and performance of deep learning: An empirical study. In *Proceedings of the Tenth ACM International Conference on Future Energy Systems*. 315–325.

[36] Qwen Team. 2025. Qwen3 Technical Report. arXiv:2505.09388 [cs.CL] https://arxiv.org/abs/2505.09388

[37] Amin Vahdat and Mark Lohmeyer. 2023. Announcing Cloud TPU v5e GA for cost-efficient AI model training and inference. https://cloud.google.com/blog/products/compute/announcing-cloud-tpu-v5e-in-ga. Google Cloud Blog.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 30. Curran Associates, Inc., 5998–6008. https://arxiv.org/abs/1706.03762

[39] Vasanth Venkatachalam and Michael Franz. 2005. Power reduction techniques for microprocessor systems. *ACM Computing Surveys (CSUR)* 37, 3 (2005), 195–237.

[40] vLLM contributors. 2023. vLLM: High-throughput and Memory-efficient Inference and Serving Engine for LLMs. https://github.com/vllm-project/vllm. Accessed: 2025-08-21.

[41] vLLM contributors. 2025. Optimization and Tuning. https://docs.vllm.ai/en/latest/configuration/optimization.html. Documentation states prefill is compute-bound and decode is memory-bound.

[42] Zibo Wang, Yijia Zhang, Fuchun Wei, Bingqiang Wang, Yanlin Liu, Zhiheng Hu, Jingyi Zhang, Xiaoxin Xu, Jian He, Xiaoliang Wang, et al. 2025. Using Analytical Performance/Power Model and Fine-Grained DVFS to Enhance AI Accelerator Energy Efficiency. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 1118–1132.

[43] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. 2024. Offline energy-optimal llm serving: Workload-based energy models for llm inference on heterogeneous systems. *ACM SIGENERGY Energy Informatics Review* 4, 5 (2024), 113–119.

[44] Yuxing Xiang, Xue Li, Kun Qian, Wenyuan Yu, Ennan Zhai, and Xin Jin. 2025. ServeGen: Workload Characterization and Generation of Large Language Model Serving in Production. arXiv:2505.09999 [cs.DC] https://arxiv.org/abs/2505.09999

[45] 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)*. 521–538.

[46] J. Yu, J. Kim, and E. Seo. 2023. Know Your Enemy To Save Cloud Energy: Energy–Performance Characterization of Machine Learning Serving. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE.

[47] C. Zhang, A. G. Kumbhare, I. Manousakis, D. Zhang, P. A. Misra, R. Assis, K. Woolcock, N. Mahalingam, B. Warrier, D. Gauthier, L. Kunnath, S. Solomon, O. Morales, M. Fontoura, and R. Bianchini. 2021. Flex: High-Availability Datacenters with Zero Reserved Power. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA '21)*.

[48] Y. Z. Zhao, D. W. Wu, and J. Wang. 2024. ALISA: Accelerating Large Language Model Inference via Sparsity-Aware KV Caching. In *Proceedings of the 51st Annual International Symposium on Computer Architecture (ISCA '24)*.

[49] Z. Zheng, X. Ren, F. Xue, Y. Luo, X. Jiang, and Y. You. 2023. Response Length Perception and Sequence Scheduling: An LLM-Empowered LLM Inference Pipeline. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*.

[50] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. 193–210.

[51] Z. Zhou, X. Wei, J. Zhang, and G. Sun. 2022. PetS: A Unified Framework for Parameter-Efficient Transformers Serving. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC '22)*. USENIX.