GreenServe: Energy-Efficient and SLO-Aware Disaggregated LLM Serving via Adaptive Frequency Control and State-Space Routing

ISCA 2026 Submission **#89** – Confidential Draft – Do NOT Distribute!!

***Abstract*—The energy cost of Large Language Model (LLM) inference is rapidly becoming a barrier to sustainable and scalable deployment. Although modern serving architectures expose distinct prefill and decode behaviors, existing systems fail to exploit these phase differences for energy-efficient serving under strict latency SLOs. This paper introduces GreenServe, the first system that explicitly targets and reduces the energy bloat in modern prefill-decode (P/D) disaggregated LLM serving. Guided by a control-theory perspective, GreenServe separates two levers: per-instance operating-point selection (GPU frequency per iteration) and system-level state-space routing of requests. We empirically observe that LLM inference exhibits a U-shaped energy-frequency curve creating "sweet spots" that depend on phase behavior and load. GreenServe exploits this by combining phase-specific, iteration-level frequency selection driven by a lightweight, online-adaptive latency predictor, with a decode state-space guided router that avoids architectural granularity-induced inefficiencies, all while meeting desired SLOs. We implement GreenServe using SGLang and evaluate it across multiple models and real-world workloads. Our results show GreenServe reduces end-to-end energy by up to 36.3% versus a static max-frequency baseline while maintaining high SLO attainment, and generalizes to newer GPUs. These results point to sustainable LLM serving via phase-aware, iteration-level frequency selection coupled with architecture-aware routing.**

I. Introduction

Large Language Models (LLMs) have become the cornerstone of modern AI services, powering applications ranging from conversational assistants [15], [29], code generation tools [3], [13], and agentic systems [53], [57]. Recent studies show that LLM inference now accounts for over 90% of total AI infrastructure utilization for major providers [9]. As these systems are deployed at massive scale, inference energy consumption has emerged as one of the most critical bottlenecks to sustainable and cost-effective AI operation. Recent industry reports indicate that even major cloud providers are constrained by power delivery limits, not just hardware availability [10], [30]. This industry-wide inflection highlights a fundamental tension between performance scaling and energy capacity: data centers can no longer rely on simply provisioning more accelerators to sustain model growth, which motivates energy-efficient LLM serving as a primary design goal.

While the conventional wisdom of dynamic frequency scaling-based energy control holds that reducing frequency directly lowers power [14], the net effect on energy (time $\times$ power) for LLM serving is not straightforward, as the increase in latency, which is crucial for user experience, varies significantly depending on workload characteristics. In particular, our empirical profiling of LLM inference reveals an interesting non-monotonic energy-frequency relationship. As shown in Fig. 1, while reducing A100 GPU frequency from 1410 MHz to 1005 MHz (-28.7%) does increase execution time, this increase is sub-linear. Consequently, the total energy follows a U-shaped curve with respect to frequency. This U-shaped trend suggests that at low frequencies, execution time dominates energy, whereas at high frequencies, power dominates; in the middle lies an energy sweet spot.

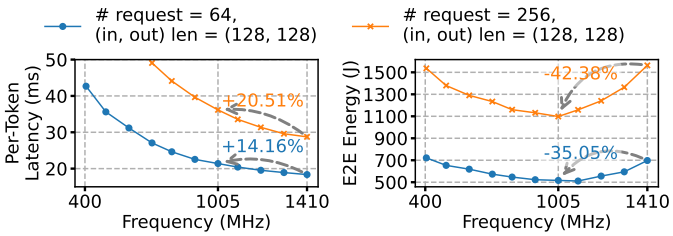


Fig. 1: The decreasing per-token latency and the U-shaped energy-frequency curve across varying GPU frequencies, for LLaMA-3.1-8B [16] served on an A100 with SGLang [59].

While this observation exposes a promising optimization opportunity, unfortunately what works well for conventional data center frequency control does not directly translate to energy-efficient LLM serving. LLM applications are often real-time and interactive, with phase-specific *Service Level Objectives* (SLOs), including *Time-To-First-Token* (TTFT) and *Inter-Token Latency* (ITL). Violating these SLOs degrades user experience and downstream system responsiveness such as agentic pipelines [57]. Modern LLM serving systems commonly employ continuous batching [56] to batch prefill and decode tokens together for higher GPU utilization. However, this coupled design introduces latency contention and phase interference [60], making it difficult to control energy without violating distinct, phase-specific SLOs. Furthermore, achieving energy savings without SLO violations is complicated, and our analysis reveals several insights: *Insight-1*: Real-world workloads exhibit strong temporal variations in request rates and types, which lead to shifting prefill-decode throughput demand and short-term workload fluctuations. For effectiveness, an energy controller must be adaptive and responsive (e.g., able to react on timescales of tens of milliseconds), but most static and power-capping methods incur coarse, high-latency adjustments (§ III-A). *Insight-2*: The energy-latency trade-off is neither uniform nor monolithic. The prefill and decode phases have different U-shaped energy-frequency relationships

and latency-frequency sensitivities. This implies that a single, coupled frequency policy is inherently suboptimal and makes it difficult to meet distinct, phase-specific SLOs (§ III-B). *Insight-3*: Energy efficiency also depends on architectural execution granularity. We observe that when decode batch sizes cross certain thresholds (e.g., 256), GPU kernel tiling shift discontinuously, leaving compute units idle and creating a "staircase" pattern in latency and energy curves (§ III-C).

To address these challenges, we build upon prefill-decode (P/D) disaggregation [31], [60] and present GreenServe, the first energy-efficient SLO-aware disaggregated LLM serving system that jointly optimizes GPU frequency and request routing under latency SLOs. Specifically, GreenServe introduces three key techniques: (1) EcoFreq (§ V-C), a responsive frequency controller that adjusts GPU frequency at fine-grained, per-iteration granularity to respond to instantaneous workload fluctuations while adhering to SLOs; (2) EcoPred (§ V-D), a lightweight yet accurate model inference time predictor, trained offline from profiling data and continuously adapted online to maintain accuracy under dynamic workloads and frequencies; (3) EcoRoute (§ V-E), an architecture-aware, state-space guided request router that dynamically steers decode requests by navigating a load-frequency state space to maximize overall energy efficiency while meeting SLOs. Together, these techniques allow GreenServe to jointly optimize for SLO attainment, energy efficiency, and dynamic load adaptation.

We implement GreenServe on SGLang [59], a production-grade LLM inference engine, and make the following contributions: **(1)** We propose the first optimization framework for energy efficient LLM inference under P/D disaggregation, which enables principled control of energy-latency trade-offs under SLO constraints. **(2)** We are the first to identify two new architectural phenomena in disaggregated LLM serving: phase-specific U-shaped energy-frequency relationships and architectural granularity-induced inefficiency, which motivate phase-specific and architecture-aware energy control. **(3)** We design and implement GreenServe, the first SLO-aware and energy-efficient disaggregated LLM serving system that incorporates phase-specific and responsive frequency controller, as well as state-space guided request routing that delivers the best energy-latency efficiency. We conduct extensive evaluations across various LLMs and workloads. Our results show that GreenServe achieves up to 36.3% energy savings while preserving TTFT and ITL SLO attainment rates.

II. Background and Related Work

A. LLM Serving Systems

Numerous systems have been proposed to improve LLM serving efficiency, including advanced batching strategies for throughput optimization [11], [56], memory management techniques such as PagedAttention [22], CPU offloading [39], [54], and vAttention [33], and GPU kernel-level optimizations for accelerating attention and decoding [2], [8], [28], [37], [50]. To better utilize available resources and improve scheduling, model parallelism and pipelining frameworks have been introduced [2], [24], [32], along with parameter sharing mechanisms to reduce memory and computation overhead [38]. Speculative decoding has emerged as a promising technique to reduce tail latency by predicting likely tokens ahead of time and verifying them in parallel [25], [42]. Additionally, systems like FastServe [52] explore preemptive scheduling policies to reduce job completion time (JCT) in multi-tenant serving environments. While these systems reduce energy consumption as a byproduct of latency or throughput improvements, GreenServe directly targets energy efficiency through frequency and routing-aware scheduling, which has been relatively underexplored despite its critical implications for sustainable deployment.

B. Energy-Efficient LLM Serving

Several works have started to explore energy-efficient and power management frameworks for LLM inference [12], [23], [35], [43]–[45]. For example, DynamoLLM [45] shows benefits of GPU frequency control in LLM inference serving based on request characteristics (predicted input/output tokens) to yield energy savings. It proposes a hierarchical design to efficiently route incoming requests to respective GPU pools that vary by model sharding and frequency while changing frequency on a coarse-grained basis of load characteristics. ThrottLL'eM [19] predicts future KV cache usage and batch size to reduce frequencies and instance sizes for energy efficiency. Similarly, μ-Serve [34] shows benefits of dynamic frequency scaling and proposes a model-serving framework to optimize for power consumption by co-serving multiple ML models. These works show early signals towards power-saving opportunities by coarse-grained control in LLM inference but do not explore the significant benefits of fine-grained control, especially in P/D disaggregated serving. Our work focuses on deeply understanding the fine-grained frequency and routing control for varying loads and SLOs, and showing the benefits of using them with P/D disaggregated LLM serving. EcoServe [23] focuses on reducing the operational and embodied carbon emissions throughout the hardware lifecycle. It proposes reusing, allocating and provisioning GPUs and CPUs based on offline/online inference, workload demand, model-size etc. TAPAS [44] focused on routing requests to specific instances within rows/aisles of GPU data-center racks by exploiting available thermal and power slack to avoid high-power hotspots. Recently, Heron [35] proposes placing GPUs closer to renewable energy sources and adjust workloads for improved efficiency. These works are complementary to our contribution and underscore the importance of sustainable AI. We deeply study frequency-control under varying loads and GreenServe is the first to explore energy-efficient LLM serving under P/D-disaggregated architecture systematically with SLO-aware feedback baked into the system.

C. P/D Disaggregation in LLM Inference

To better manage the different compute characteristics of the prefill and decode phases, recent work has proposed prefill-decode (P/D) disaggregation, which assigns the two phases

to separate GPU nodes. Systems such as SplitWise [31], TetriInfer [18], Llumnix [46] and DistServe [60] show that this separation improves goodput and TTFT and ITL SLO attainment rates by avoiding phase-level contention and enabling distinct execution policies. As such, major open-source LLM inference libraries such as vLLM [22] and SGLang [59] have also added runtime support for this architectural pattern. However, these efforts focus primarily on improving metrics like latency and throughput. To our knowledge, they have neither explored the energy implications of P/D disaggregation nor exploited the new opportunities that this architectural separation enables for energy optimization, such as phase-specific fine-grained frequency control and energy-aware P/D routing policies.

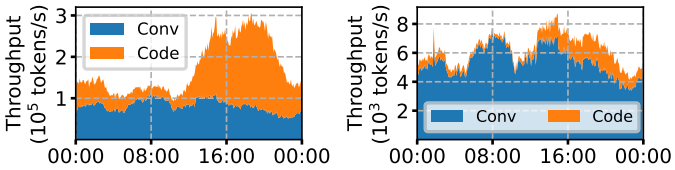
III. Observations and Opportunities

In this section, we characterize the performance-energy behavior of LLaMA-3.1-8B [16] served by SGLang [59] on an A100 GPU. We profile key metrics, including TTFT, ITL, and end-to-end (E2E) energy consumption (measured via NVIDIA Management Library [27]). Furthermore, we analyze real-world workload traces, the Azure LLM Inference Trace 2024 dataset [45], to identify temporal and batching patterns.

A. Multi-Timescale Workload Dynamics in LLM Serving

Real-world LLM workloads exhibit variation at two distinct timescales. At a *coarse-grained* level, request arrival rates and prompt lengths fluctuate significantly over time, leading to shifting prefill-decode throughput demand and motivating energy adaptation. Taking the Azure LLM Inference Trace 2024 dataset [45] as an example, it categorizes requests into two types: *conversation* and *code*. As shown in Fig. 2a, the prefill throughput of conversation requests remains relatively stable over a day, while code requests exhibit a significant diurnal pattern, peaking in the afternoon and evening. Furthermore, code requests typically have shorter decode lengths. Consequently, as depicted in Fig. 2b, the overall variation of the decode throughput is much smaller than that of prefill. At a *fine-grained* level, the number of active tokens and requests in prefill changes rapidly over time, as shown in Fig. 3, due to dynamic batching, request completion, and various request lengths, which create short-term load fluctuations that directly affect GPU utilization and latency. These P/D demand variations motivate phase-specific and workload-aware energy adaptation. Most importantly, these fast dynamics also make static or coarse adjustments ineffective, highlighting the need for more responsive frequency control for energy efficiency.

Insight #1: Real-world LLM inference workloads vary across two timescales: coarse-grained shifts in prefill-decode demand and fine-grained, iteration-level fluctuations in prefill batch composition. These multi-timescale dynamics make static or coarse frequency control policies ineffective, motivating a phase-specific and responsive approach for energy-efficient LLM serving with SLO guarantees.



(a) Prefill throughput. **(b) Decode throughput.**

Fig. 2: Temporal variations of prefill and decode throughput in Azure LLM Inference Trace 2024 dataset. Conversation and code requests exhibit distinct diurnal patterns. Colors are stacked.

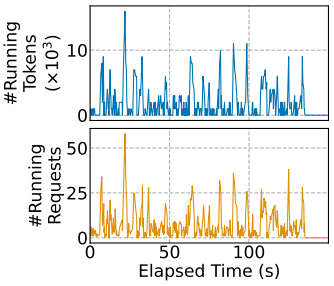


Fig. 3: Rapid iteration-level workload fluctuations in the number of running requests and tokens of the prefill phase.

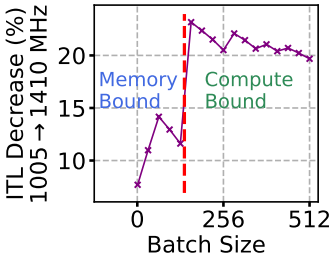


Fig. 4: ITL decrease percentage from frequency scaling (1005 → 1410 MHz) for decode phase under various batch sizes.

B. U-Shaped Energy-Frequency Relationship and Sensitivities in P/D Disaggregated LLM Serving

Frequency scaling is a common technique for energy optimization, yet its effect on LLM inference is non-monotonic. Our measurements in Fig. 1 reveal a clear U-shaped energy-frequency relationship, which indicates an energy-efficiency sweet spot. To understand how this behavior manifests under P/D disaggregation, we examine the two phases separately.

Fig. 5 shows the energy and latency characteristics of the two phases across various GPU frequencies, using requests with varied input and output lengths for each phase. Both phases show monotonically decreasing latency-frequency curves and U-shaped energy-frequency curves, with 1005 MHz consistently being the energy-optimal point before a sharp energy upsurge towards the maximum frequency (1410 MHz). Frequencies below 1005 MHz are strictly suboptimal, as they increase both energy and latency. However, the two phases differ markedly in shape and sensitivity, stemming from fundamental architectural factors.

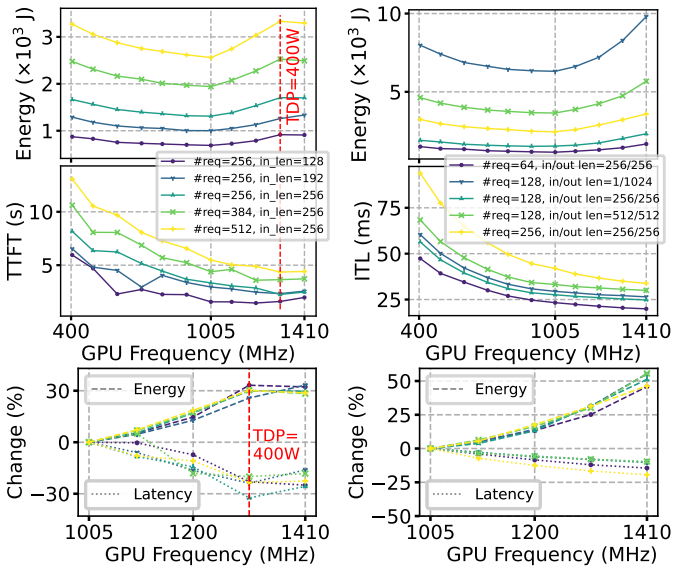
According to [14], the power P of CMOS circuits (i.e., the physical foundation of GPUs) comprises a frequency-irrelevant static part and a frequency-dependent dynamic part:

$$P = P_{\text{static}} + P_{\text{dynamic}} \approx P_{\text{static}} + CV^2 f \sim A + Bf^{1+\alpha}, \quad (1)$$

where A, B, C, α are coefficients, V is voltage, which also increases with frequency. We denote this change as $V^2 \sim f^{\alpha}$, where $\alpha > 0$ and can increase with f. Thus, the execution time T and energy consumption E can be expressed as:

$$T \sim f^{-\beta} \sim [(P - A)/B]^{-\frac{\beta}{1+\alpha}}, \quad (2)$$

$$E = PT \sim Af^{-\beta} + Bf^{1+\alpha-\beta}, \quad (3)$$



(a) Prefill phase. **(b) Decode phase.**

Fig. 5: Impact of GPU frequency on energy (top) and latency (middle) for P/D phases on A100 with LLaMA-3.1-8B. The bottom shows the energy and latency relative changes by frequency increases from 1005 MHz to 1410 MHz. Prefill hits TDP limitations near 1305 MHz. Both phases exhibit U-shaped energy-frequency curves with 1005 MHz as energy-optimal points, but they differ significantly in shape and sensitivity.

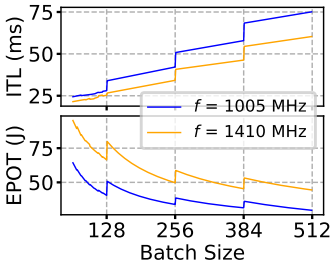


Fig. 6: Batch size boundary and "staircase" effects in ITL and EPOT in the decode phase.

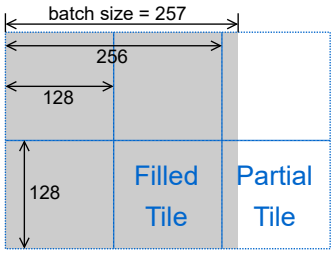


Fig. 7: The tile quantization effect of GEMM. Tile size is 128, and request batch size is 257.

where $0 \leq \beta \leq 1$ is an arithmetic intensity-related coefficient: $\beta = 1$ indicates fully compute-bound, and $\beta = 0$ indicates fully memory-bound. The E-f plot of Eq. 3 is a U-shaped curve, as the $Af^{-\beta}$ term dominates at low frequencies and the $Bf^{1+\alpha-\beta}$ term dominates at high frequencies. This theoretical analysis matches our empirical observation in Fig. 5. Furthermore, for the compute-bound prefill phase, $\beta \approx 1$, which means it benefits almost proportionally from higher frequency (Eq. 2) and gets significant TTFT reduction at the cost of higher energy (Fig. 5a bottom), quickly driving the GPU toward its Thermal Design Power (TDP) limit at ≈1305 MHz. In contrast, for the common memory-bound decode phase, $\beta < 1$, which means decode derives a sub-linear benefit from higher frequency (Eq. 2), as Fig. 5b bottom shows: increasing f from 1005 MHz to 1410 MHz yields only ≈20% ITL reduction at the cost of ≈50% higher energy. Notably, as batch size grows and arithmetic intensity increases, decode gradually transitions toward compute-bound behavior and responds more strongly to higher frequencies (Fig. 4).

Insight #2: While both P/D phases exhibit U-shaped energy-frequency relationships and the corresponding energy sweet spots, they differ remarkably in shape and sensitivity. Given that lowering frequency inevitably increases latency, these distinct characteristics motivate an adaptive, phase-aware policy to minimize energy while preserving SLOs.

C. Architectural Granularity-Induced Energy Inefficiency

Modern LLM serving systems batch multiple requests to improve GPU utilization [56]. However, our profiling reveals that implicit batch size boundaries create architecturally induced inefficiencies in both energy and latency, especially during the decode phase. As shown in Fig. 6, while increasing batch sizes generally reduces *Energy-Per-Output-Token* (EPOT) and improves efficiency, this trend is periodically disrupted: crossing certain thresholds (e.g., 256 → 257) causes abrupt performance drops, resulting in a "staircase" pattern in ITL and EPOT. A similar but weaker effect exists in prefill, which is masked by its larger token counts (see Appx. A).

This discontinuity arises from tile quantization. GPUs execute GEMM by partitioning the output matrix into fixed-size "tiles" and assigning them to thread blocks [26]. This "staircase" effect occurs when the batch size is not an exact multiple of the tile dimensions. For example, as Fig. 7 shows, when the batch size increases from 256 to 257, the GPU must launch an entirely new, but almost empty, tile (e.g., only 6.25% useful data) to process this single extra request. This under-utilized tile requires the same execution time (duration) as a fully utilized tile. This leads to an increase in the total number of tiles, which in turn causes a sudden jump in latency (ITL) and a drop in efficiency (EPOT), forming the "staircase" seen in Fig. 6. While coupled serving architectures (e.g., chunked prefill [1]) can mask this behavior by merging prefill and decode tokens into shared kernels, P/D disaggregated architecture amplifies the problem, as the decode instances operate with smaller, highly variable batch sizes.

Insight #3: GPU architectural granularity, specifically SM-level tiling and scheduling boundaries, causes discontinuous "staircase" effects in energy and latency as batch sizes cross certain thresholds. This hardware-level phenomenon is amplified in P/D disaggregated LLM inference, and reveals an overlooked source of energy inefficiency and motivates architecture-aware frequency and batching control.

IV. Problem Formulation

We consider an LLM serving system that follows the P/D disaggregation architecture. The system contains N_P prefill instances (indexed by $p = 1, \cdots, N_P$) and N_D decode instances (indexed by $d = 1, \cdots, N_D$), each running independently. Incoming requests must first complete prefill on one of the prefill instances and subsequently complete decode on one of the decode instances. Let S_P and S_D denote the TTFT and ITL SLO targets for these two phases. The objective is to minimize the end-to-end energy consumption across all instances while meeting phase-specific SLOs: prefill TTFT $< S_P$, decode

ITL $< S_D$. This essentially yields the following constrained optimization problem: minimize total energy of all iterations across all instances, subject to satisfying TTFT and ITL SLO constraints for each request.

V. System Design

Motivated by the insights in § III and the formulation in § IV, we propose GreenServe, an LLM serving system built on a P/D disaggregation architecture to achieve SLO-aware and energy-efficient inference. The following subsections detail the design of GreenServe.

A. Design Principle: A Control Theory Perspective

The global optimization problem in § IV defines the objective of minimizing end-to-end energy under per-phase latency SLO constraints. However, solving the problem exactly is combinatorial, and an NP-hard assignment problem [36], which is computationally intractable. Moreover, the future request arrivals are unknown at runtime. To make this problem tractable, we turn to a control theory perspective. Instead of solving the global problem monolithically, we observe that a P/D-disaggregated system naturally exposes two complementary control dimensions: (i) Instance-level execution configuration, which determines the latency-energy tradeoff of each prefill and decode instance. This is analogous to a local control loop in classical control system [51]: each subsystem adjusts its internal operation point (e.g., frequency) based on its current measurable state. (ii) Assignment of incoming requests, which determines how load is distributed across instances. This corresponds directly to *state-space control* in multi-subsystem systems [20], [21]: routing a request shifts an instance from one region of its state space to another, which affects both its future latency and its required operating point.

These connections are not merely conceptual. In control theory terms, instance configuration control is a continuous, fine-grained control problem over a single subsystem, while routing is a discrete state-space navigation problem over multiple subsystems whose state jointly determine the global energy-latency envelop of the system. This mapping guides our design: GreenServe applies instance-level control to continuously select energy-efficient operating points, while global routing navigates the state spaces of instances to avoid high-energy regions such as architectural tile boundaries.

B. Overall Architecture

Fig. 8 illustrates the overall architecture of GreenServe, which integrates three co-designed components that collectively deliver SLO-aware, energy-efficient LLM serving.

EcoFreq (§ V-C): A lightweight, phase-specific, and iteration-level frequency controller for both prefill and decode instances based on current system state and batch information, maximizing energy efficiency while preserving SLO attainment. Its frequency selection is guided by the inference time predictor EcoPred.

EcoPred (§ V-D): A lightweight yet accurate prefill and decode model inference time predictor for both prefill and decode. Built on XGBoost [5], it combines both offline profiling and online adaptation to maintain accuracy under workload dynamics and mitigates distribution shift between offline and online data.

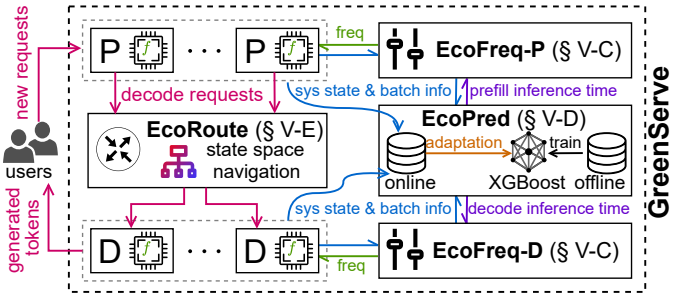


Fig. 8: Overall architecture of GreenServe. New requests from users are first handled by prefill instances (round-robin) to generate the first output token. EcoRoute then dispatches the requests to decode instance, from which generated tokens are streamed back to the users. EcoFreq runs in a separate process to control the frequency of each P/D instance. EcoPred provides prefill and decode model inference time predictions based on system states and batch information, while also collecting these metrics from all instances for online adaptation.

EcoRoute (§ V-E): An energy-efficient router that assigns prefill-generated requests to decode instances. It mitigates architectural granularity-induced energy inefficiency (§ III-C) by analyzing and navigating each decode instance state spaces before making routing decisions, avoiding unnecessary transition into high-frequency regions.

C. EcoFreq: SLO-Aware and Iteration-Level Frequency Control for Responsive Energy Adaptation

Although modern GPUs (e.g., NVIDIA A100/H100) expose APIs to set frequencies, they do not tune frequencies dynamically in response to load [41]. As such, using them effectively for SLO-aware and energy-efficient LLM serving is non-trivial due to two challenges. First, temporal variations in P/D demands (§ III-A) require distinct frequency policies for each phase, as they exhibit different optimal operating points depending on the workload. The inevitable energy-latency trade-off (§ III-B) also requires a control policy that adapts to real-time workload variations while guaranteeing SLO attainment. Second, coarse-grained frequency adjustment is ineffective in handling rapid iteration-level dynamism in workloads (§ III-A), yet existing fine-grained frequency control introduces expensive overhead. Recent studies report ∼50 ms latency per change via blocking `nvidia-smi` commands [45], which is unsuitable for latency-sensitive engine loops where each generation step typically completes within tens to hundreds of milliseconds.

To overcome these challenges, we design EcoFreq, a responsive, SLO-aware controller that performs iteration-level GPU frequency adjustment for both prefill and decode instances. Fig. 9 illustrates how EcoFreq interacts with P/D instances. To overcome the overhead of frequency settings via `nvidia-smi`, EcoFreq introduces multiple optimizations. It runs as a dedicated process outside the critical inference path, communicating with the inference engine via ZeroMQ [4] for low-overhead communication. Rather than expensively setting

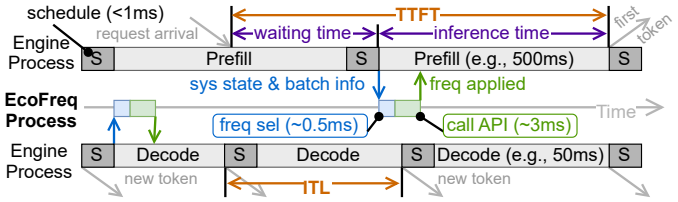


Fig. 9: Execution flow of EcoFreq with P/D instances. EcoFreq runs as a separate lightweight process that collects system states and batch from the engine and performs sub-millisecond iteration-level frequency control.

frequency via `nvidia-smi`, EcoFreq leverages `nvmlDeviceSetGpuLockedClocks` API from NVIDIA Management Library [27] to apply new frequency. Our measurements show it consumes <3 ms. Upon scheduling a new batch, the inference engine sends its current batch information $\mathbf{B}$ and system state $\mathbf{M}$ to EcoFreq. EcoFreq then selects an appropriate GPU frequency and applies it immediately. This per-iteration loop enables responsive adaptation to instantaneous workload conditions. Furthermore, the frequency selection algorithm (we will discuss it later) executes in ∼0.5 ms. The total end-to-end overhead of EcoFreq per iteration is thus <4 ms and is overlapped with the engine's model inference. Consequently, although the beginning of an inference step may use the previous iteration's frequency, this period is negligible compared to the total model inference time (e.g., 500 ms for prefill and 50 ms for decode), introducing negligible control inaccuracy. This lightweight design achieves per-iteration responsiveness to workload variations with negligible overhead, enabling EcoFreq to apply fine-grained, independent control for each phase and instance, thereby maximizing energy savings while ensuring SLO attainment.

Alg. 1 details the SLO-aware, phase-specific frequency selection algorithm of EcoFreq. Let $\mathcal{F}$ be the set of frequency options. Each invocation of EcoFreq comprises three steps: ❶ **Queue Check** (Line 1-3): If there are requests in *waiting* state (queued requests awaiting scheduling), EcoFreq selects the highest frequency $\max(\mathcal{F})$ to promptly clear waiting requests and prevent SLO violations due to queue buildup. This prevents a scenario where a lower frequency, while sufficient for *running* (actively executing decode steps) request SLOs, would reduce overall throughput, causing waiting requests to accumulate and subsequently violate their SLOs. ❷ **Phase-Specific Adjustment** (Line 4-8): The target token latency for each phase can be decomposed as:

$$\text{Latency} = T_{\text{fixed}} + T_{\text{inference}}(f) \leq \text{SLO} \tag{4}$$

where T_{fixed} is frequency-*irrelevant* time (e.g., request scheduling and waiting) and $T_{\text{inference}}(f)$ is the frequency-*dependent* GPU model inference time. As shown in Fig. 9, T_{fixed} differs by phase. For *decode*, T_{fixed} is negligible scheduling time (<1ms). Conversely, for *prefill*, T_{fixed} is primarily the request's waiting time (i.e., from arrival to being scheduled for running), which is non-negligible. Since frequency scaling only controls $T_{\text{inference}}(f)$, EcoFreq sets a phase-adjusted target S

Algorithm 1 EcoFreq: SLO-Aware Frequency Selection

Require: Current system status $\mathbf{M}$, batch information $\mathbf{B}$, frequency option list $\mathcal{F}$, model inference time predictors $T_P(\cdot)$ and $T_D(\cdot)$, TTFT SLO S_P, ITL SLO S_D.
Ensure: Selected GPU frequency f.
1: **if** HASWAITINGREQS($\mathbf{M}$) **then** ▷ **step ❶ Queue check**
2: **return** $\max(\mathcal{F})$ ▷ clear backlogged reqs timely
3: **end if**
4: **if** ISPREFILL($\mathbf{B}$) **then** ▷ **step ❷ Phase adjustment**
/* prefill: deduct the waiting time from SLO budget */
5: $S \leftarrow S_P - \max(\mathbf{T}_{\text{waiting}}(\mathbf{B}))$, $T_{\text{inference}}(\cdot) \leftarrow T_P(\cdot)$
6: **else**
7: $S \leftarrow S_D$, $T_{\text{inference}}(\cdot) \leftarrow T_D(\cdot)$ ▷ /* decode */
8: **end if**
9: **for** $f \in \text{sorted}(\mathcal{F})$ **do** ▷ **step ❸ Frequency selection**
10: **if** $T_{\text{inference}}(\mathbf{M}, \mathbf{B}, f) \leq S$ **then**
11: **return** f ▷ minimum frequency to meet the SLO
12: **end if**
13: **end for**
14: **return** $\max(\mathcal{F})$ ▷ no freq can meet SLO, return max

for $T_{\text{inference}}(f)$ to ensure SLO attainment:

$$T_{\text{inference}}(f) \leq S = \begin{cases} S_P - \max\left(\mathbf{T}_{\text{waiting}}\right), & \text{prefill,} \\ S_D, & \text{decode,} \end{cases} \tag{5}$$

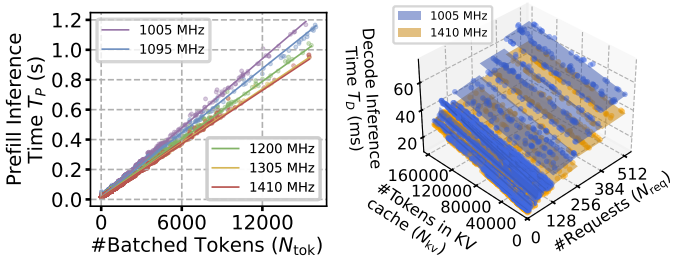
where S_P and S_D are TTFT and ITL SLOs, respectively, and $\max\left(\mathbf{T}_{\text{waiting}}\right)$ is the maximum waiting time among requests within a prefill batch. ❸ **Frequency Selection** (Line 9-14): Using the prefill ($T_P(\cdot)$) or decode ($T_D(\cdot)$) inference time predictors, EcoFreq iterates through candidate frequencies $\mathcal{F}$ in ascending order, choosing the lowest frequency that makes $T_{\text{inference}}(f) \leq S$. If none meets the target, it defaults to $\max(\mathcal{F})$ to preserve SLO attainment.

The predictors $T_P(\cdot)$ and $T_D(\cdot)$ take ∼0.5 ms to execute each time. We achieve this through careful model selection (discussed in § V-D), which enables per-iteration frequency selection of EcoFreq.

D. EcoPred: Online-Adaptive, Load-Aware Frequency–Latency Prediction for Iteration-Level Control

The effectiveness of EcoFreq relies on the accurate model inference time predictors $T_P(\cdot)$ and $T_D(\cdot)$ under varying system states and GPU frequencies. To achieve this, we design online-adaptive predictors EcoPred built on XGBoost [5], a model widely adopted for its ability to accurately model DL operator performance while remaining lightweight and robust to feature interactions [6], [7]. EcoPred combines *offline profiling* with *online adaptation* to remain accurate under dynamic workloads.

Rationale: predictability and feature selection. For offline analysis, we perform the same profiling as in § III-B but collect additional system state metrics: number of running requests (i.e., batch size) N_{req}, batched token number N_{tok}, and token number in KV cache storage N_{kv}. Fig. 10 shows an example of profiling results for LLaMA-3.1-8B on A100.



(a) Prefill. (b) Decode.

Fig. 10: Offline profiling results for LLaMA-3.1-8B on A100. Each point represents a collected sample. Prefill latency grows near linearly with batched tokens, and decode latency exhibits tile-structured behavior across request count and KV-cache size. The lines and planes are visual guides to illustrate predictability.

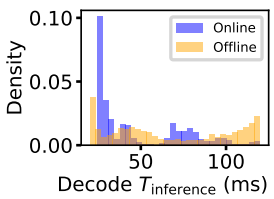


Fig. 11: Distribution shift of Decode model inference time between online data and offline data.

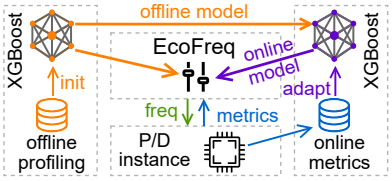


Fig. 12: Overview of EcoPred. The model is initialized by offline profiling and continuously refined using online metrics, enabling accurate runtime model inference time prediction.

The results clearly demonstrate that model inference time exhibits strong predictability under varying GPU frequencies and system states. For prefill, inference time shows a strong linear relationship with N_{tok}. Conversely, decode inference time exhibits the "staircase" effect (as discussed in § III-C), yet remains highly predictable from $(N_{\text{req}}, N_{\text{kv}})$ within each tile. These observations are formalized as:

$$T_P(\mathbf{M}, \mathbf{B}, f) \approx a_f \cdot N_{\text{tok}} + b_f, \tag{6}$$

$$T_D(\mathbf{M}, \mathbf{B}, f) \approx c_f \cdot N_{\text{req}} + d_f \cdot N_{\text{kv}} + e_f \quad \text{(each tile)}. \tag{7}$$

This predictability reflects the distinct computational characteristics of two phases. For the compute-bound prefill phase, total computation theoretically scales linearly with the batched token number N_{tok} [8], [49], yielding the highly linear relationship in Fig. 10a. In contrast, for the decode phase, N_{req} determines the GEMM computational load [49], while N_{kv} dictates the Attention layer computation and KV cache memory access [8]. This profiling also captures the transition of decode from being memory-bound to compute-bound (as shown in Fig. 4). As Fig. 10b illustrates, the latency gap between 1005 MHz and 1410 MHz is small for small batch sizes N_{req}, but widens significantly when batch sizes are larger as the workload becomes more computation-intensive.

Based on these observations, we build EcoPred based on XGBoost [5] to capture the predictability and create accurate models for $T_P(\cdot)$ and $T_D(\cdot)$:

$$T_P(\mathbf{M}, \mathbf{B}, f) = \text{XGBoost}(f, N_{\text{tok}}), \tag{8}$$

$$T_D(\mathbf{M}, \mathbf{B}, f) = \text{XGBoost}(f, N_{\text{req}}, N_{\text{kv}}). \tag{9}$$

To ensure this offline model can be applied across diverse online serving situations, we profile uniformly over the feasible ranges of N_{tok}, N_{req}, and N_{kv}. This broad, distribution-agnostic sampling avoids bias toward any particular workload pattern and provides coverage of the model's execution, while online adaptation later corrects for any distribution shift.

Online adaptation to distribution shift. While offline profiling provides broad coverage, its data distribution can differ from real serving deployments due to dynamic request arrivals and varying sequence lengths. Fig. 11 plots the distributions of sampled online and offline data for a decode instance, revealing a clear distribution shift. While offline profiling often uses a uniform data distribution to minimize model bias, online data does not adhere to this assumption due to the dynamic nature of request arrivals and lengths.

To overcome such distribution shift, we propose *online adaptation*. As shown in Fig. 12, before the deployment, EcoPred first performs the offline profiling and model training to build the initial offline predictors $T_P^{\text{offline}}(\cdot)$ and $T_D^{\text{offline}}(\cdot)$. During runtime, EcoPred continuously collects online performance metrics and initiates a background finetuning process every X new samples to produce the updated finetuned models $T_P^{\text{online}}(\cdot)$ and $T_D^{\text{online}}(\cdot)$. This online finetuning allows EcoPred to maintain accuracy despite the dynamic workload variations.

At runtime, EcoFreq queries EcoPred to estimate P/D model inference times. Since the models are lightweight, this prediction incurs negligible overhead (e.g., <0.5 ms).

E. EcoRoute: State Space-Guided Routing

While EcoFreq adaptively controls GPU frequency for energy efficiency, request routing offers another key opportunity for energy optimization in multi-instance systems.

State-space perspective. As established earlier in § V-D, the ITL and energy consumption of a decode instance are functions of the triple $(N_{\text{req}}, N_{\text{kv}}, f)$. Since EcoFreq determines f based on $(N_{\text{req}}, N_{\text{kv}})$, each instance's operating condition can be represented as a point in the $(N_{\text{req}}, N_{\text{kv}})$ state space. When requests arrive, finish, or generate new tokens, the instance moves to a different point in this state space.

A visualization of this state space (Fig. 13) shows the "frequency cliff" created by batch-size boundaries (§ III-C). When N_{req} crosses a batch size boundary (e.g., 256), ITL increases significantly, which forces EcoFreq to raise the frequency sharply to maintain SLOs. This produces a disproportionate jump in energy consumption. Therefore, an effective routing policy should try to keep instances away from these high-cost regions whenever possible.

Motivating example. Consider 2 decode instances, 520 total requests, and a batch boundary at 256. A "symmetric" router (red point in Fig. 13) dispatching 260 requests to each instance forces both to cross the "cliff", compelling higher frequency to meet the SLO. In contrast, an "asymmetric" routing decision (blue points in Fig. 13) could assign 240 requests to one instance, keeping it below the "cliff" at a low frequency, while the other instance handles 280 requests, incurring a small, SLO-compliant ITL increase. Although slightly imbalanced, this assignment is more energy efficient overall. This illustrates

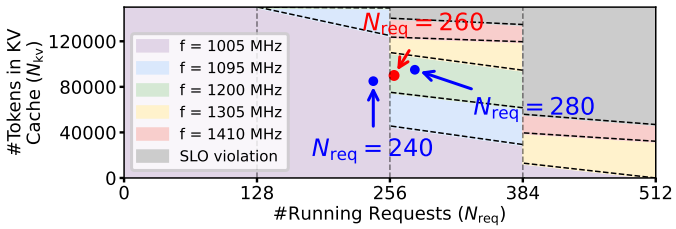


Fig. 13: Example state space of a decode instance. The axes represent the number of running requests (N_{req}) and the number of tokens in KV cache (N_{kv}). Colored regions show the operating frequency f chosen by EcoFreq. The annotated points illustrate how different routing choices (e.g., $N_{\text{req}} = 240, 260, 280$) place the instance in low- or high-frequency regions.

why routing should consider where each instance lies in the state space, rather than treating all instances uniformly.

State-space guided routing design. We propose EcoRoute to exploit this opportunity. Instead of exhaustively searching for the optimal assignment, EcoRoute makes online routing decision when each decode request arrives. Leveraging the concept of state space, EcoRoute performs a "what-if" analysis: it hypothetically assigns the request to *all* instances, navigates their state spaces and evaluates frequency changes, then makes the routing decision. Alg. 2 details the algorithm, and Fig. 14 provides an illustration for a 2-instance system. The algorithm compares the *current* frequencies $\mathcal{F}$ with the hypothetically *resulting* frequencies $\mathcal{F}'$, leading to the following cases:

❶ **No frequency changes** ($\mathcal{F} = \mathcal{F}'$). There are two sub-cases: **(a)** If all instances share the same frequency, the algorithm falls back to a round-robin policy. **(b)** Otherwise, it selects the instance with lowest current frequency ($\min(\mathcal{F})$).

❷ **Partial instances increase the frequencies**. Let Δ be a frequency difference threshold. There are two sub-cases: **(a)** If the difference between lowest and highest resulting frequency is within the threshold ($\max(\mathcal{F}') - \min(\mathcal{F}') \leq \Delta$), EcoRoute selects the instance with the lowest *unchanged* frequency. This rule avoids unnecessarily crossing the previously discussed frequency "cliff". **(b)** Otherwise, EcoRoute selects the instance with the lowest resulting frequency ($\min(\mathcal{F}')$). This rule serves as a safeguard against severe load imbalance.

❸ **All instances increase the frequencies**. EcoRoute selects the instance with lowest resulting frequency ($\min(\mathcal{F}')$).

Fig. 15 conceptually illustrates how EcoRoute obtains energy benefits over a round-robin policy. Round-robin suffers from the frequency "cliff" problem at batch size boundaries, like the "symmetric" router discussed before. In contrast, EcoRoute is an "asymmetric" router, avoiding unnecessary frequency increases near the batch size boundaries. Consequently, one instance (orange) operates for significantly more time at the energy-efficient low frequency than under the round-robin.

While EcoRoute is effective for decode instances, we apply a simple round-robin policy for prefill instances for two reasons. First, the prefill phase does not exhibit the significant batch size boundary effect seen in decode, as its batched token numbers are typically large (§ III-C and Appx. A). Second, the prefill phase is largely stateless, showing little system status correlation between consecutive batches (discussed in Fig. 3).

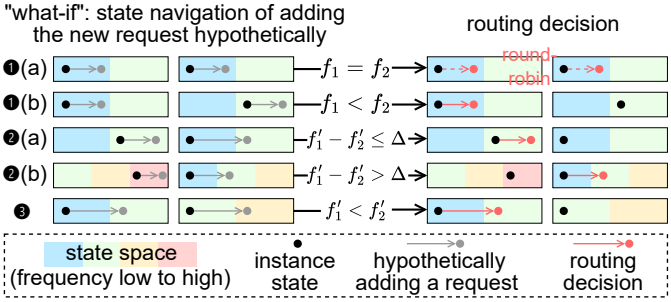


Fig. 14: Illustration of EcoRoute for a system with 2 decode instances. Colors indicate the state space regions (mapped to different frequencies by EcoFreq). The initial instance frequencies are f_1 and f_2, while f_1' and f_2' are the resulting frequencies after hypothetically adding the new request. EcoRoute evaluates these outcomes and selects the routing decision based on the corresponding decision rule (❶a, ❶b, ❷a, ❷b, or ❸).

Algorithm 2 EcoRoute: Decode Request Routing

Require: Decode instance list $\mathcal{D} = \{d_i\}_{i=1}^n$ and their state list $\mathcal{M} = \{m_i\}_{i=1}^n$, request for routing r, frequency decision of EcoFreq $\text{freq}(\cdot)$, frequency threshold Δ.

Ensure: Selected decode instance $d \in \mathcal{D}$.

$\mathcal{F} = \{\text{freq}(m_i)\}_{i=1}^n$ ▷ current freq of all instances
$\mathcal{F}' = \{\text{freq}(m_i \oplus r)\}_{i=1}^n$ ▷ frequency after hypothetically adding request r, $\oplus$ denotes adding request
if $\mathcal{F} = \mathcal{F}'$ **then** ▷ **case ❶**: no frequency changes
  if AllSame($\mathcal{F}$) **then** ▷ **❶(a)**: all freqs are same
    return RoundRobin($\mathcal{D}$) ▷ round-robin
  else ▷ **❶(b)**: not all same
    return $d_{\text{argmin}_i \mathcal{F}}$ ▷ min current freq
  end if
else if PartialChanged($\mathcal{F}$, $\mathcal{F}'$) **then** ▷ **case ❷**: some frequencies increase
  if $\max(\mathcal{F}') - \min(\mathcal{F}') \leq \Delta$ **then** ▷ **❷(a)**: diff $\leq \Delta$
    return $d_{\text{argmin}_i \text{Unchanged}(\mathcal{F},\mathcal{F}')}$ ▷ min unchanged
  else ▷ **❷(b)**: diff $> \Delta$
    return $d_{\text{argmin}_i \mathcal{F}'}$ ▷ min resulting freq
  end if
end if
return $d_{\text{argmin}_i \mathcal{F}'}$ ▷ **case ❸**: all frequency increase

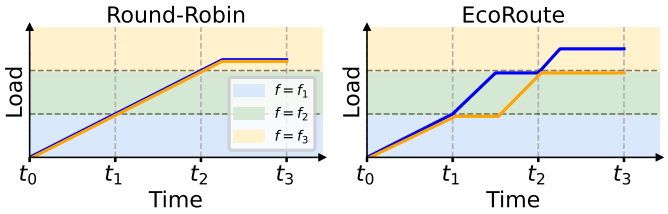


Fig. 15: Conceptual comparison of round-robin and EcoRoute with two decoding instances. Line colors represent different instances. Shaded regions show the frequency levels required to meet SLOs ($f_1 < f_2 < f_3$). Under EcoRoute, one instance (orange) remains in a lower-frequency region for longer, yielding better energy efficiency than round-robin.

Consequently, the state-space navigation approach used by EcoRoute is inapplicable to prefill.

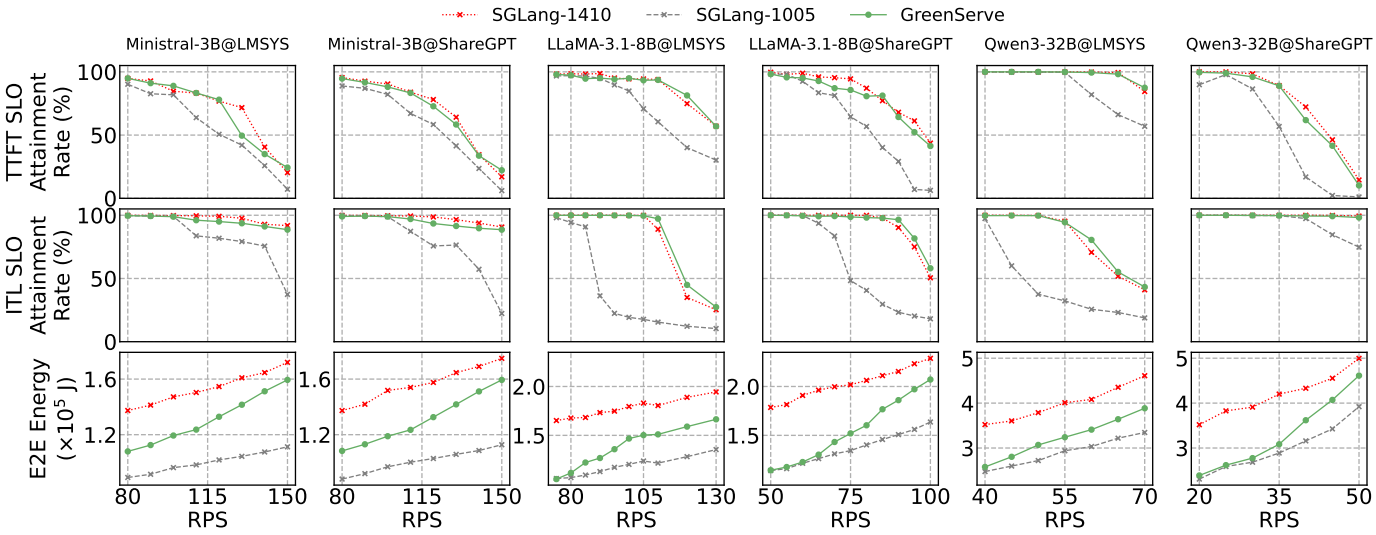


Fig. 16: The TTFT/ITL SLO attainment rates and E2E energy consumption comparing GreenServe and baselines. GreenServe consistently maintains TTFT and ITL SLO attainment rates comparable or slightly better comparing to SGLang-1410, while significantly reducing E2E energy consumption across diverse models and datasets.

VI. Evaluation

A. Evaluation Methodology

Implementation. We implement GreenServe in Python on top of SGLang (v0.4.7.post1) [59], a widely used LLM inference framework with production-level P/D disaggregation support. As described in § V-C, EcoFreq runs in a separate process to overlap overhead, and pyNVML [17] is used for minimal-overhead frequency setting. Appx. B details the XGBoost model configurations of EcoPred.

Models and Workloads. We evaluate GreenServe on three models: Ministral-3B [47], LLaMA-3.1-8B [16] and Qwen3-32B [55]. We use the ShareGPT [48] and LMSYS-Chat-1M [58] datasets as workloads, with request arrival intervals following a Poisson distribution at controlled request arrival rates (quantified by *Requests-Per-Second*, RPS). Length characteristics of datasets are detailed in Appx. C.

Metrics. We evaluate key performance and energy-efficiency metrics of GreenServe. For performance, we focus on TTFT and ITL, which align with user-facing quality-of-service expectations. To quantify the effectiveness under these constraints, we report the SLO attainment rate, defined as the percentage of requests satisfying the SLO thresholds for both TTFT and ITL. For energy efficiency, we examine end-to-end (E2E) energy consumption via pyNVML and report in Joules.

Hardware Testbeds. All experiments are performed on NVIDIA A100-80G SXM4 GPUs connected via NVLink.

B. Main Result

We evaluate GreenServe across all models and datasets, using a 2P2D (2 prefill and 2 decode instances) configuration. For Qwen3-32B, 2-way Tensor Parallelism [40] is used for each instance. The frequency options of EcoFreq are set to $\mathcal{F} = \{1005, 1410\}$ MHz, and the imbalance prevention threshold of EcoRoute is set to $\Delta = 500$, which allows both frequencies to co-exist for different instances. TTFT/ITL SLOs are set to 200/20, 600/60, 1200/120 ms for Ministral-3B, LLaMA-3.1-8B, and Qwen3-32B, respectively. While DynamoLLM [45] and throttLL'eM [19] are the most relevant systems to our work, they are not included for comparison due to the lack of publicly available code. As such, we construct two baselines based on SGLang: **SGLang-1005**, using a static 1005 MHz frequency (the "sweet spot" described in § III-B), and **SGLang-1410**, using the static maximum frequency (1410 MHz, the default strategy of A100 [41]). These baselines use SGLang's default round-robin request routing. For each configuration, we report: (1) TTFT SLO attainment rate, (2) ITL SLO attainment rate, and (3) E2E energy consumption.

Fig. 16 shows our results, from which we have two key findings. First, GreenServe consistently achieves comparable TTFT SLO attainment rates and comparable or slightly better ITL SLO attainment rate compared to SGLang-1410, which operates at the static maximum frequency. It significantly outperforms SGLang-1005, which is energy efficient but yields substantially lower SLO attainment. This ability to maintain high SLO attainment is attributed to the SLO-aware design of EcoFreq. It adaptively selects the GPU frequency for each engine iteration to ensure the model inference time remains within the budget. Notably, the slightly better ITL SLO attainment compared to SGLang-1410 in some situations is attributed to EcoRoute. As illustrated in Fig. 13, EcoRoute allows one instance to avoid crossing a batch size boundary and frequency "cliff", at the cost of a marginal ITL increase in another instance. This increase is outweighed by the ITL reduction benefit from avoiding the boundary. Second, while preserving latency SLOs, GreenServe reduces E2E energy consumption by up to 36.3% compared to SGLang-1410. Its effectiveness is most pronounced at low request rates, where a static low frequency is sufficient for SLO targets,

so GreenServe dynamically operates at a low frequency most of the time. Conversely, at high request rates, it dynamically increases frequency to maintain SLO attainment, resulting in smaller yet still significant energy benefits. We also provide the throughput metrics in Appx. D and latency Cumulative Distribution Functions (CDFs) in Appx. E for more details.

Overall, these improvements validate that SLO-aware frequency control and request routing effectively optimize energy consumption while preserving the quality of service.

C. Analysis Results

How does each module of GreenServe bring benefits? To quantify the contribution of each component in GreenServe, we conduct experiments similar to § VI-B using the ShareGPT dataset and LLaMA-3.1-8B model, but with two additional system configurations: **EcoFreq-only** and a full version of **GreenServe** (EcoFreq + EcoRoute). Fig. 17 presents the results. First, EcoFreq-only achieves substantial energy savings for both prefill and decode instances compared to the static high-frequency baseline SGLang-1410, validating the effectiveness of its SLO-aware frequency control policy. Fig. 18 provides an example of real-time frequency traces. At low request rates, both P/D instances mainly operate at low frequency to conserve energy, while the proportion of high-frequency time increases for higher request rates due to SLO targets. Second, the full GreenServe achieves additional energy reductions compared to EcoFreq-only, which is decode specific as EcoRoute is only applied to the decode instances. Fig. 19 compares round-robin and EcoRoute by the batch size and frequencies of the two decode instances over time. It shows EcoRoute can restrict the batch size of one instance under a specific boundary (256) so it can operate at a lower frequency.

How does GreenServe perform under different latency SLO targets? We evaluate GreenServe on LLaMA-3.1-8B with the ShareGPT dataset, following the settings in § VI-B but testing three different TTFT/ITL SLO profiles: 400/40 ("low"), 600/60 ("medium"), and 800/80 ("high"). Fig. 20 presents the results. Across all SLO profiles, GreenServe consistently achieves SLO attainment rates comparable or slightly better than the static maximum frequency baseline. Specifically, as the SLO constraints are relaxed (i.e., from "low" to "high"), GreenServe's adaptive frequency control trades higher latency for lower energy consumption. This demonstrates the latency-energy trade-off, providing the flexibility to define custom scenario and phase-specific SLOs, such as prioritizing low ITL for responsiveness or low energy consumption to reduce cost. This result further emphasizes the necessity of the phase-specific design as discussed in § III.

How important is it to have per-iteration frequency control? In § III-A, we show rapid iteration-level workload variation in LLM serving, which necessitates the design for per-iteration frequency control. GreenServe provides this iteration-level responsiveness, unlike prior approaches [45] that adjust frequency at coarser, window-based intervals (e.g., 5s). To verify its benefit, we adapt EcoFreq to operate at various fixed time intervals, evaluating it using the ShareGPT dataset on

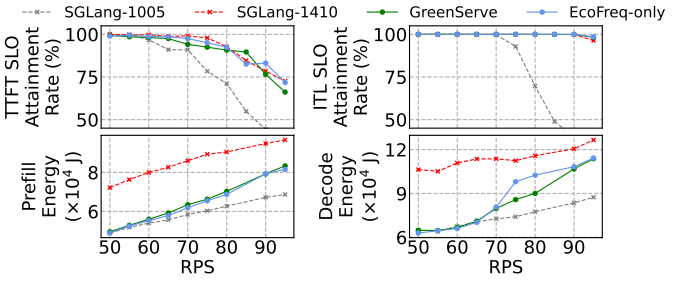


Fig. 17: Comparison of EcoFreq-only and full GreenServe. EcoFreq-only reduces energy for both prefill and decode while preserving SLOs, and EcoRoute yields additional energy savings for the decode phase by avoiding high-frequency regions.

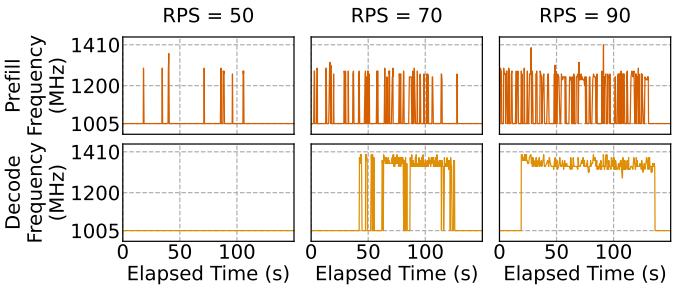


Fig. 18: Real-time frequency traces of EcoFreq-only for P/D instances at different request rates. At low RPS, instances primarily operate at low frequency, As RPS increases, both shift to high frequency more often to maintain SLOs.

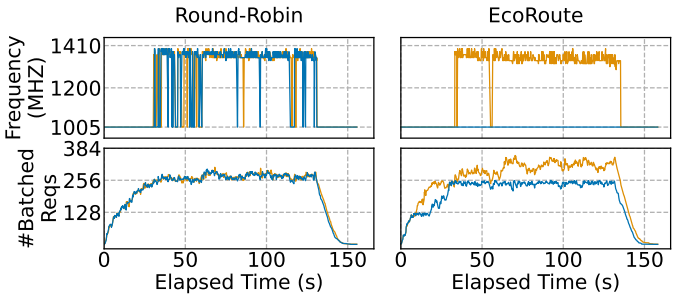


Fig. 19: An example of how EcoRoute differs from round-robin policy (RPS=75). Each line represents a decode instance. EcoRoute keeps one instance below the batch size boundary (256), allowing it to run at a lower frequency for longer time.

LLaMA-3.1-8B under the 1P1D configuration (where EcoRoute has no effect). The results in Fig. 21 demonstrate that window-based frequency control degrades SLO attainment for both phases. Specifically, per-iteration frequency responsiveness is particularly critical for the prefill phase, where window-based control shows larger degradation. Due to dynamic batching, the total number of tokens processed in a prefill batch (and thus the optimal frequency) can vary significantly from one iteration to the next, as Fig. 3 shows. A window-based approach is too coarse-grained to react to such rapid changes. In contrast, as shown in Fig. 19, the load of the decode instance shows smoother change in batched requests, leading to relative insensitivity to window interval settings.

Do more frequency levels bring benefits? In § VI-B, GreenServe is configured with two-level frequency options.

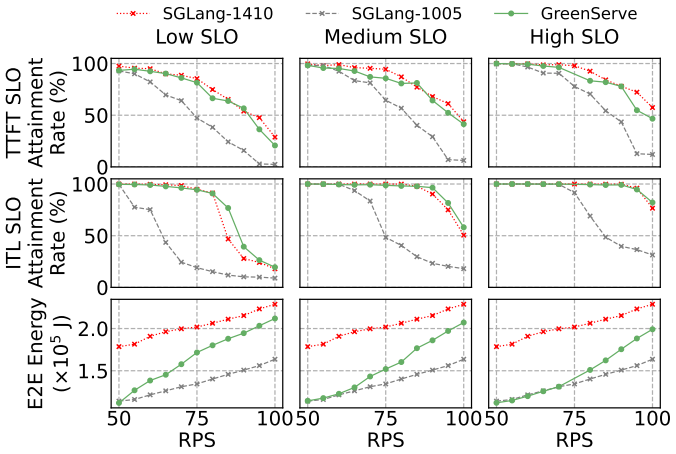


Fig. 20: Comparison of GreenServe and baselines under different SLO profiles (low, medium, high). Across all settings, GreenServe maintains strong TTFT and ITL SLO attainment rates while providing significant energy savings.

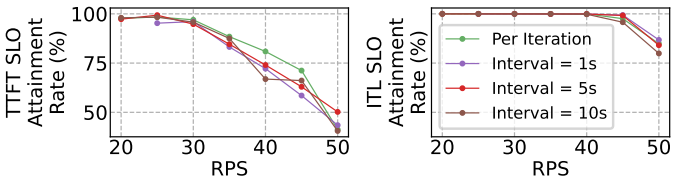


Fig. 21: Latency SLO attainment rates across different frequency control intervals. Our per-iteration design provides the best responsiveness, and thereby the highest SLO attainment rates.

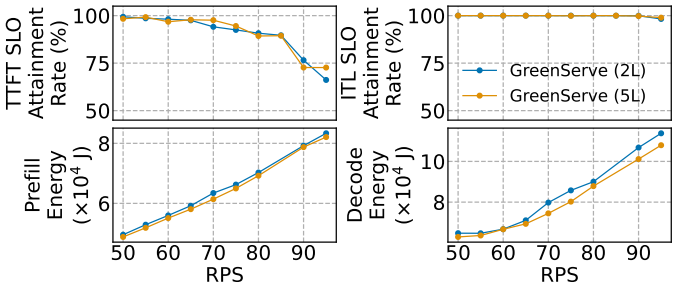


Fig. 22: Comparison of GreenServe with different frequency levels. GreenServe (2L) uses two frequency levels, while GreenServe (5L) uses five. Both configurations maintain similar TTFT/ITL attainment and energy trends when combined with EcoRoute.

To analyze the potential benefits of finer-grained levels, we extend EcoFreq's frequency options to five levels: $\mathcal{F} = \{1005, 1095, 1200, 1305, 1410\}$ MHz, and evaluate it using the ShareGPT dataset on LLaMA-3.1-8B, following the same settings as § VI-B. Fig. 22 presents the results. For prefill, finer frequency granularity yields negligible benefits, as the significant iteration-level workload variation (Fig. 3) makes the middle levels largely unused. In contrast, the decode reveals a trade-off: finer granularity offers slight energy savings but at the cost of lower SLO attainment rates. This allows users to select appropriate granularities based on their specific performance-energy priorities.

Is EcoPred accurate? Can online adaptation improve accuracy? To verify these aspects, we compare EcoPred's predicted values against the empirical latency results from § VI-B, evaluating both the initial offline-trained and the online-adapted models. Fig. 23 shows the results. Both TTFT and ITL predictions consistently achieve low mean absolute errors (MAEs) across all models, which enables the SLO-aware frequency selection of EcoFreq. Furthermore, online adaptation provides a clear MAE improvement, which successfully mitigates the sample distribution shift (Fig. 11).

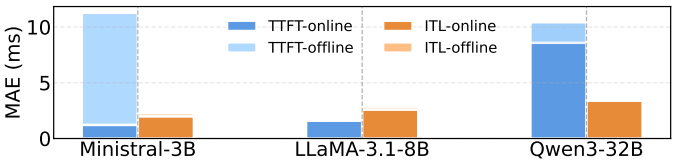


Fig. 23: Mean absolute errors (MAEs) of EcoPred for both TTFT and ITL, comparing the offline-only model to the online-adapted model across all evaluated models.

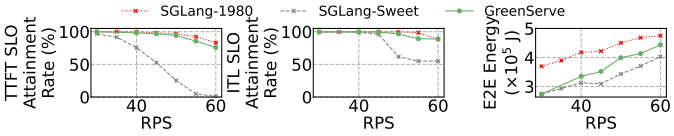


Fig. 24: Latency SLO attainment and E2E energy consumption of GreenServe on GH200, using Qwen3-32B and ShareGPT dataset.

Can GreenServe generalize to other hardware? To verify GreenServe's generalizability, we evaluate it on NVIDIA GH200 Grace Hopper Superchips with Qwen3-32B (without Tensor Parallelism) and ShareGPT dataset. GH200 exhibits similar but different U-shaped energy-frequency curves compared to A100: the energy sweet spot is 1095 MHz for prefill and 1395 MHz for decode, with a maximum frequency of 1980 MHz (more details in Appx. F). Accordingly, we set EcoFreq's frequency options as $\mathcal{F}_P = \{1095, 1980\}$ MHz and $\mathcal{F}_D = \{1395, 1980\}$ MHz. The baselines are **SGLang-1980** (fixed maximum frequency) and **SGLang-Sweet** (P/D instances fixed at their respective sweet spots). Other settings follow § VI-B. Fig. 24 shows the results, which leads to a similar conclusion as on A100: GreenServe maintains comparable latency SLO attainment and substantial energy savings, demonstrating its generalizability to different hardware.

VII. Conclusions

In this work, we present GreenServe, the first system for energy-efficient and SLO-aware LLM inference under prefill-decode (P/D) disaggregation. GreenServe exploits the fundamentally different compute and memory behaviors of the two phases to introduce phase-specific, iteration-level frequency control and architecture-aware, state-space navigation-based routing. Together, these techniques enable fine-grained control of latency-energy that existing serving systems cannot achieve. Through comprehensive evaluation on state-of-the-art LLMs and real-world datasets, GreenServe achieves up to 36.3% energy savings while preserving latency SLO attainment. These results highlight the potential of fine-grained, phase-aware co-design in advancing sustainable LLM serving systems.

REFERENCES

[1] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. Gulavani, A. Tumanov, and R. Ramjee, "Taming {Throughput-Latency} tradeoff in {LLM} inference with {Sarathi-Serve}," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 117–134.

[2] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley, and Y. He, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[3] Anysphere, "Cursor: The best way to code with ai," 2025, [accessed 2025-11-08]. [Online]. Available: https://cursor.com/

[4] T. Z. authors, "Zeromq," 2025, [accessed 2025-11-11]. [Online]. Available: https://zeromq.org/

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[6] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and K. Arvind, "{TVM}: An automated {End-to-End} optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[7] T. Chen, L. Zheng, E. Yan, Z. Jiang, T. Moreau, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Learning to optimize tensor programs," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[8] T. Dao, "Flashattention-2: Faster attention with better parallelism and work partitioning," *arXiv preprint arXiv:2307.08691*, 2023.

[9] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustainable Computing: Informatics and Systems*, vol. 38, p. 100857, 2023.

[10] C. Elsworth, K. Huang, D. Patterson, I. Schneider, R. Sedivy, S. Goodman, B. Townsend, P. Ranganathan, J. Dean, A. Vahdat, B. Gomes, and J. Manyika, "Measuring the environmental impact of delivering ai at google scale," *arXiv preprint arXiv:2508.15734*, 2025.

[11] J. Fang, Y. Yu, C. Zhao, and J. Zhou, "Turbotransformers: an efficient gpu serving system for transformer models," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 389–402.

[12] J. Fernandez, C. Na, V. Tiwari, Y. Bisk, S. Luccioni, and E. Strubell, "Energy considerations of large language model inference and efficiency optimizations," *arXiv preprint arXiv:2504.17674*, 2025.

[13] Github, "Github copilot · your ai pair programmer," 2024, [accessed 2025-11-08]. [Online]. Available: https://github.com/features/copilot

[14] R. Gonzalez, B. M. Gordon, and M. A. Horowitz, "Supply and threshold voltage scaling for low power cmos," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1210–1216, 1997.

[15] Google, "Google gemini," 2025, [accessed 2025-11-08]. [Online]. Available: https://gemini.google.com/app

[16] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson, A. Spataru, B. Roziere, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Wyatt, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Guzmán, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Thattai, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. Kloumann, I. Misra, I. Evtimov, J. Zhang, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Prasad, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, K. El-Arini, K. Iyer, K. Malik, K. Chiu, K. Bhalla, K. Lakhotia, L. Rantala-Yeary, L. van der Maaten, L. Chen, L. Tan, L. Jenkins, L. Martin, L. Madaan, L. Malo, L. Blecher, L. Landzaat, L. de Oliveira, M. Muzzi, M. Pasupuleti, M. Singh, M. Paluri, M. Kardas, M. Tsimpoukelli, M. Oldham, M. Rita, M. Pavlova, M. Kambadur, M. Lewis, M. Si, M. K. Singh, M. Hassan, N. Goyal, N. Torabi, N. Bashlykov, N. Bogoychev, N. Chatterji, N. Zhang, O. Duchenne, O. Çelebi, P. Alrassy, P. Zhang, P. Li, P. Vasic, P. Weng, P. Bhargava, P. Dubal, P. Krishnan, P. S. Koura, P. Xu, Q. He, Q. Dong, R. Srinivasan, R. Ganapathy, R. Calderer, R. S. Cabral, R. Stojnic, R. Raileanu, R. Maheswari, R. Girdhar, R. Patel, R. Sauvestre, R. Polidoro, R. Sumbaly, R. Taylor, R. Silva, R. Hou, R. Wang, S. Hosseini, S. Chennabasappa, S. Singh, S. Bell, S. S. Kim, S. Edunov, S. Nie, S. Narang, S. Raparthy, S. Shen, S. Wan, S. Bhosale, S. Zhang, S. Vandenhende, S. Batra, S. Whitman, S. Sootla, S. Collot, S. Gururangan, S. Borodinsky, T. Herman, T. Fowler, T. Sheasha, T. Georgiou, T. Scialom, T. Speckbacher, T. Mihaylov, T. Xiao, U. Karn, V. Goswami, V. Gupta, V. Ramanathan, V. Kerkez, V. Gonguet, V. Do, V. Vogeti, V. Albiero, V. Petrovic, W. Chu, W. Xiong, W. Fu, W. Meers, X. Martinet, X. Wang, X. Wang, X. E. Tan, X. Xia, X. Xie, X. Jia, X. Wang, Y. Goldschlag, Y. Gaur, Y. Babaei, Y. Wen, Y. Song, Y. Zhang, Y. Li, Y. Mao, Z. D. Coudert, Z. Yan, Z. Chen, Z. Papakipos, A. Singh, A. Srivastava, A. Jain, A. Kelsey, A. Shajnfeld, A. Gangidi, A. Victoria, A. Goldstand, A. Menon, A. Sharma, A. Boesenberg, A. Baevski, A. Feinstein, A. Kallet, A. Sangani, A. Teo, A. Yunus, A. Lupu, A. Alvarado, A. Caples, A. Gu, A. Ho, A. Poulton, A. Ryan, A. Ramchandani, A. Dong, A. Franco, A. Goyal, A. Saraf, A. Chowdhury, A. Gabriel, A. Bharambe, A. Eisenman, A. Yazdan, B. James, B. Maurer, B. Leonhardi, B. Huang, B. Loyd, B. D. Paola, B. Paranjape, B. Liu, B. Wu, B. Ni, B. Hancock, B. Wasti, B. Spence, B. Stojkovic, B. Gamido, B. Montalvo, C. Parker, C. Burton, C. Mejia, C. Liu, C. Wang, C. Kim, C. Zhou, C. Hu, C.-H. Chu, C. Cai, C. Tindal, C. Feichtenhofer, C. Gao, D. Civin, D. Beaty, D. Kreymer, D. Li, D. Adkins, D. Xu, D. Testuggine, D. David, D. Parikh, D. Liskovich, D. Foss, D. Wang, D. Le, D. Holland, E. Dowling, E. Jamil, E. Montgomery, E. Presani, E. Hahn, E. Wood, E.-T. Le, E. Brinkman, E. Arcaute, E. Dunbar, E. Smothers, F. Sun, F. Kreuk, F. Tian, F. Kokkinos, F. Ozgenel, F. Caggioni, F. Kanayet, F. Seide, G. M. Florez, G. Schwarz, G. Badeer, G. Swee, G. Halpern, G. Herman, G. Sizov, Guangyi, Zhang, G. Lakshminarayanan, H. Inan, H. Shojanazeri, H. Zou, H. Wang, H. Zha, H. Habeeb, H. Rudolph, H. Suk, H. Aspegren, H. Goldman, H. Zhan, I. Damlaj, I. Molybog, I. Tufanov, I. Leontiadis, I.-E. Veliche, I. Gat, J. Weissman, J. Geboski, J. Kohli, J. Lam, J. Asher, J.-B. Gaya, J. Marcus, J. Tang, J. Chan, J. Zhen, J. Reizenstein, J. Teboul, J. Zhong, J. Jin, J. Yang, J. Cummings, J. Carvill, J. Shepard, J. McPhie, J. Torres, J. Ginsburg, J. Wang, K. Wu, K. H. U, K. Saxena, K. Khandelwal, K. Zand, K. Matosich, K. Veeraraghavan, K. Michelena, K. Li, K. Jagadeesh, K. Huang, K. Chawla, K. Huang, L. Chen, L. Garg, L. A, L. Silva, L. Bell, L. Zhang, L. Guo, L. Yu, L. Moshkovich, L. Wehrstedt, M. Khabsa, M. Avalani, M. Bhatt, M. Mankus, M. Hasson, M. Lennie, M. Reso, M. Groshev, M. Naumov, M. Lathi, M. Keneally, M. Liu, M. L. Seltzer, M. Valko, M. Restrepo, M. Patel, M. Vyatskov, M. Samvelyan, M. Clark, M. Macey, M. Wang, M. J. Hermoso, M. Metanat, M. Rastegari, M. Bansal, N. Santhanam, N. Parks, N. White, N. Bawa, N. Singhal, N. Egebo, N. Usunier, N. Mehta, N. P. Laptev, N. Dong, N. Cheng, O. Chernoguz, O. Hart, O. Salpekar, O. Kalinli, P. Kent, P. Parekh, P. Saab, P. Balaji, P. Rittner, P. Bontrager, P. Roux, P. Dollar, P. Zvyagina, P. Ratanchandani, P. Yuvraj, Q. Liang, R. Alao, R. Rodriguez, R. Ayub, R. Murthy, R. Nayani, R. Mitra, R. Parthasarathy, R. Li, R. Hogan, R. Battey, R. Wang, R. Howes, R. Rinott, S. Mehta, S. Siby, S. J. Bondu, S. Datta, S. Chugh, S. Hunt, S. Dhillon, S. Sidorov, S. Pan, S. Mahajan, S. Verma, S. Yamamoto, S. Ramaswamy, S. Lindsay, S. Lindsay, S. Feng, S. Lin, S. C. Zha, S. Patil, S. Shankar, S. Zhang, S. Zhang, S. Wang, S. Agarwal, S. Sajuyigbe, S. Chintala, S. Max, S. Chen, S. Kehoe, S. Satterfield, S. Govindaprasad, S. Gupta, S. Deng, S. Cho, S. Virk, S. Subramanian, S. Choudhury, S. Goldman, T. Remez, T. Glaser, T. Best, T. Koehler, T. Robinson, T. Li, T. Zhang, T. Matthews, T. Chou, T. Shaked, V. Vontimitta, V. Ajayi, V. Montanez, V. Mohan, V. S. Kumar, V. Mangla, V. Ionescu, V. Poenaru, V. T. Mihailescu, V. Ivanov, W. Li, W. Wang, W. Jiang, W. Bouaziz, W. Constable, X. Tang, X. Wu, X. Wang, X. Wu, X. Gao, Y. Kleinman, Y. Chen, Y. Hu, Y. Jia, Y. Qi, Y. Li, Y. Zhang, Y. Zhang, Y. Adi, Y. Nam, Yu, Wang, Y. Zhao, Y. Hao, Y. Qian, Y. Li, Y. He, Z. Rait, Z. DeVito, Z. Rosnbrick, Z. Wen, Z. Yang, Z. Zhao, and Z. Ma, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[17] A. Hirschfeld, "nvidia-ml-py · pypi," 2025, [accessed 2025-11-08]. [Online]. Available: https://pypi.org/project/nvidia-ml-py/

[18] C. Hu, H. Huang, L. Xu, X. Chen, J. Xu, S. Chen, H. Feng, C. Wang, S. Wang, Y. Bao, N. Sun, and Y. Shan, "Inference without interference: Disaggregate llm inference for mixed downstream workloads," *arXiv preprint arXiv:2401.11181*, 2024.

[19] A. K. Kakolyris, D. Masouros, P. Vavaroutsos, S. Xydis, and D. Soudris, "throttll'em: Predictive gpu throttling for energy efficient llm inference serving," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025.

[20] R. E. Kalman, "On the general theory of control systems," in *Proceedings first international conference on automatic control, USSR*, 1960.

[21] E. Kreindler, "Contributions to the theory of time-optimal control," *Journal of the Franklin Institute*, vol. 275, no. 4, pp. 314–344, 1963.

[22] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th symposium on operating systems principles*, 2023, pp. 611–626.

[23] Y. Li, Z. Hu, E. Choukse, R. Fonseca, G. E. Suh, and U. Gupta, "Ecoserve: Designing carbon-aware ai inference systems," *arXiv preprint arXiv:2502.05043*, 2025.

[24] Z. Li, L. Zheng, Y. Zhong, V. Liu, Y. Sheng, X. Jin, Y. Huang, Z. Chen, H. Zhang, J. E. Gonzalez, and I. Stoica, "{AlpaServe}: Statistical multiplexing with model parallelism for deep learning serving," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 663–679.

[25] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi, C. Shi, Z. Chen, D. Arfeen, R. Abhyankar, and Z. Jia, "Specinfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 932–949.

[26] NVIDIA, "Matrix multiplication background user's guide - nvidia docs," 2023, [accessed 2025-11-16]. [Online]. Available: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

[27] ——, "Nvidia management library (nvml) — nvidia developer," 2025, [accessed 2025-11-11]. [Online]. Available: https://developer.nvidia.com/management-library-nvml

[28] ——, "Nvidia/fastertransformer: Transformer related optimization, including bert, gpt," 2025, [accessed 2025-11-17]. [Online]. Available: https://github.com/NVIDIA/FasterTransformer

[29] OpenAI, "Chatgpt," 2025, [accessed 2025-11-08]. [Online]. Available: https://chatgpt.com/

[30] P. Patel, E. Choukse, C. Zhang, Í. Goiri, B. Warrier, N. Mahalingam, and R. Bianchini, "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*, 2024, pp. 207–222.

[31] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[32] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," *Proceedings of machine learning and systems*, vol. 5, pp. 606–624, 2023.

[33] R. Prabhu, A. Nayak, J. Mohan, R. Ramjee, and A. Panwar, "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*, 2025, pp. 1133–1150.

[34] H. Qiu, W. Mao, A. Patke, S. Cui, S. Jha, C. Wang, H. Franke, Z. Kalbarczyk, T. Başar, and R. K. Iyer, "Power-aware deep learning model serving with {μ-Serve}," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 75–93.

[35] T. R. Reddy, Palak, R. Gandhi, A. Parayil, C. Zhang, M. Shepperd, L. Yu, J. Mohan, S. Iyengar, S. Kalyanaraman, and D. Bhattacherjee, "Ai greenferencing: Routing ai inferencing to green modular data centers with heron," *arXiv preprint arXiv:2505.09989*, 2025.

[36] G. T. Ross and R. M. Soland, "A branch and bound algorithm for the generalized assignment problem," *Mathematical programming*, vol. 8, no. 1, pp. 91–103, 1975.

[37] J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, "Flashattention-3: Fast and accurate attention with asynchrony and low-precision," *Advances in Neural Information Processing Systems*, vol. 37, 2024.

[38] Y. Sheng, S. Cao, D. Li, C. Hooper, N. Lee, S. Yang, C. Chou, B. Zhu, L. Zheng, K. Keutzer, J. E. Gonzalez, and I. Stoica, "S-lora: Serving thousands of concurrent lora adapters," *arXiv preprint arXiv:2311.03285*, 2023.

[39] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[40] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[41] M. Špet'ko, O. Vysockỳ, B. Jansík, and L. Říha, "Dgx-a100 face to face dgx-2—performance, power and thermal behavior evaluation," *Energies*, 2021.

[42] M. Stern, N. Shazeer, and J. Uszkoreit, "Blockwise parallel decoding for deep autoregressive models," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[43] J. Stojkovic, E. Choukse, C. Zhang, I. Goiri, and J. Torrellas, "Towards greener llms: Bringing energy-efficiency to the forefront of llm inference," *arXiv preprint arXiv:2403.20306*, 2024.

[44] J. Stojkovic, C. Zhang, Í. Goiri, E. Choukse, H. Qiu, R. Fonseca, J. Torrellas, and R. Bianchini, "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*, 2025.

[45] J. Stojkovic, C. Zhang, Í. Goiri, J. Torrellas, and E. Choukse, "Dynamollm: Designing llm inference clusters for performance and energy efficiency," in *2025 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2025.

[46] B. Sun, Z. Huang, H. Zhao, W. Xiao, X. Zhang, Y. Li, and W. Lin, "Llumnix: Dynamic scheduling for large language model serving," in *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, 2024, pp. 173–191.

[47] M. A. team, "Un ministral, des ministraux — mistral ai," 2024, [accessed 2025-11-17]. [Online]. Available: https://mistral.ai/news/ministraux

[48] S. Team, "Sharegpt: Share your wildest chatgpt conversations with one click." 2025, [accessed 2025-11-17]. [Online]. Available: https://sharegpt.com/

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[50] X. Wang, Y. Xiong, Y. Wei, M. Wang, and L. Li, "Lightseq: A high performance inference library for transformers," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, 2021, pp. 113–120.

[51] N. Wiener, *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press, 2019.

[52] B. Wu, Y. Zhong, Z. Zhang, S. Liu, F. Liu, Y. Sun, G. Huang, X. Liu, and X. Jin, "Fast distributed inference serving for large language models," *arXiv preprint arXiv:2305.05920*, 2023.

[53] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversations," in *First Conference on Language Modeling*, 2024.

[54] Y. Xu, Z. Mao, X. Mo, S. Liu, and I. Stoica, "Pie: Pooling cpu memory for llm inference," *arXiv preprint arXiv:2411.09317*, 2024.

[55] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng, M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[56] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[57] Y. Zhang, Z. Ma, Y. Ma, Z. Han, Y. Wu, and V. Tresp, "Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 22, 2025, pp. 23 378–23 386.

[58] L. Zheng, W.-L. Chiang, Y. Sheng, T. Li, S. Zhuang, Z. Wu, Y. Zhuang, Z. Li, Z. Lin, E. P. Xing, J. E. Gonzalez, I. Stoica, and H. Zhang,

"Lmsys-chat-1m: A large-scale real-world llm conversation dataset," *arXiv preprint arXiv:2309.11998*, 2023.

[59] L. Zheng, L. Yin, Z. Xie, C. L. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng, "Sglang: Efficient execution of structured language model programs," *Advances in neural information processing systems*, vol. 37, pp. 62 557–62 583, 2024.

[60] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "{DistServe}: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 193–210.

APPENDIX

A. Batch Size Boundaries of Prefill Phase

Fig. 25, a zoomed-in view of Fig. 10a, focuses on the prefill phase with small numbers of batched tokens to show the relationship between prefill model inference time and batched token numbers (N_{tok}). We can observe similar "staircase-like" tile effect like decode phase (Fig. 10b). However, this effect only exist when batched token numbers are relative small. When number of batched tokens goes over about 2000 (which is the dominate situation), this effect gradually becomes less significant.

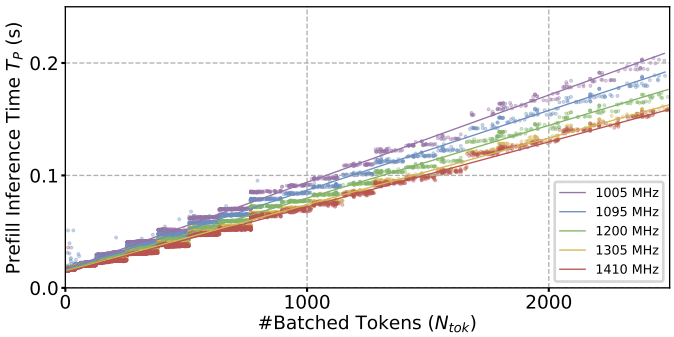


Fig. 25: Zoomed-in version of Fig. 10a for small batched token numbers: relationship between prefill model inference time and batched token numbers (N_{tok}) (LLaMA-3.1-8B on A100).

B. XGBoost Configuration of EcoPred

As discussed in § V-D, we employ XGBoost [5] to predict prefill and decode model inference times based on system states and batch information. The model configuration details are as follows:

```python
import xgboost as xgb

# prefill model
model_prefill = xgb.XGBRegressor(
    objective="reg:absoluteerror",
    n_estimators=100000,
    learning_rate=0.5,
    booster='gblinear',
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    tree_method='hist',
    random_state=42,
    eval_metric='mae',
    early_stopping_rounds=1000,
)

# decode model
model_decode = xgb.XGBRegressor(
    objective="reg:absoluteerror",
    n_estimators=1000,
    learning_rate=0.1,
    booster='gbtree',
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    tree_method='hist',
    random_state=42,
    eval_metric='mae',
    early_stopping_rounds=200,
)
```

C. Length Distribution of ShareGPT and LMSYS Dataset

Tab. I summarizes the statistical properties of the ShareGPT and LMSYS datasets used in the evaluation, including the mean and standard deviation of their input and output lengths. The cumulative distribution functions (CDFs) for these length distributions are visualized in Fig. 26. Analysis of this data shows that:

- ShareGPT has longer prefill and decode length compared to LMSYS dataset.
- ShareGPT has larger prefill/decode length ratio.

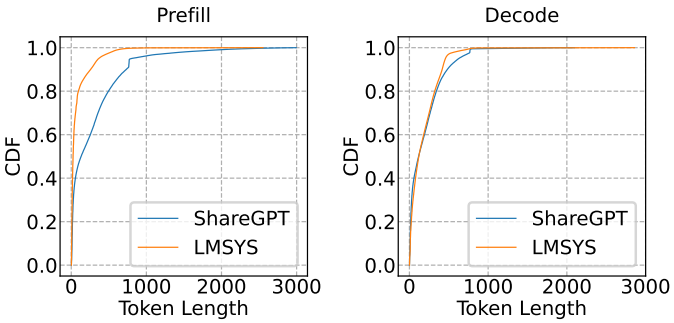


Fig. 26: CDF of prefill and decode lengths for the ShareGPT and LMSYS datasets.

TABLE I: Length distribution of datasets used in our evaluation. Std refers to standard deviation.

Dataset	Prefill		Decode	
	Mean (ms)	Std (ms)	Mean (ms)	Std (ms)
ShareGPT	280.27	375.58	190.90	209.15
LMSYS	78.40	133.29	174.57	166.13

D. Extra Throughput Data of The Main Result

We compute the throughput from the experimental results in § VI-B using ShareGPT dataset on LLaMA-3.1-8B. As shown in Fig. 27, GreenServe achieves slightly lower throughput than SGLang with static maximum frequency due to its opportunistic frequency scaling, but not at the cost of SLO violations. Furthermore, at high request rates where greater throughput is required, GreenServe can also adaptively achieve nearly the same throughput as SGLang with static maximum frequency.

E. Extra TTFT and ITL CDFs of The Main Result

Fig. 30 shows the TTFT and ITL CDFs corresponding to the results in § VI-B. The figure provides CDFs across all model and dataset combinations (organized by column). For each combination, CDFs are presented for both low request arrival rates (rows 1 and 3) and high request rates (rows 2 and 4). The results show that when the request arrival rate is low, GreenServe achieves latency CDFs similar to SGLang-1005, as the low frequency is sufficient for SLO attainment. In contrast, when the request rate is high, GreenServe adaptively increases its frequency, achieving latency CDFs similar to SGLang-1410.

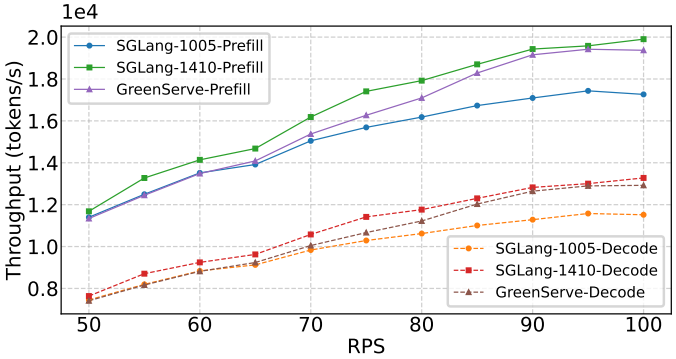
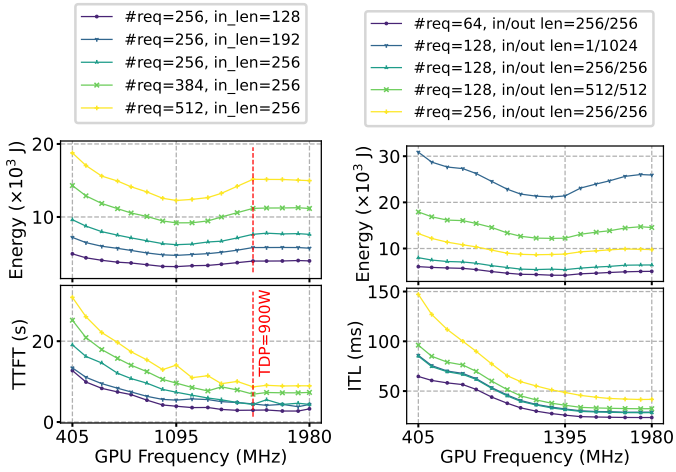


Fig. 27: Comparing throughput of GreenServe with baselines for LLaMA-3.1-8B and ShareGPT dataset.

F. U-Shape Figures of GH200

Fig. 28 illustrates the U-shaped energy-frequency curves and monotonically decreasing latency-frequency curves when serving Qwen3-32B [55] on a GH200. These results exhibit trends similar to those of the A100 (Fig. 5), but with distinct hardware-specific differences. The prefill phase remains power-intensive, hitting the GH200's 900W TDP limitation at ≈1600 MHz. Furthermore, the two phases possess different energy sweet spots: the optimal point for prefill is ≈1095 MHz, while for decode, it is ≈1395 MHz.



(a) Prefill phase. **(b) Decode phase.**

Fig. 28: Impact of GPU frequency on energy (up) and latency (down) for P/D phases on GH200 with Qwen3-32B. Prefill hits TDP limitations (900 W) near 1600 MHz. Both phases exhibit U-shaped energy-frequency curves, but prefill reach energy sweet spot ≈1095 MHz, while decode reach energy sweet spot ≈1395 MHz.

G. What happens in varying the P/D throughput demand ratio?

As discussed in § III-A, the temporal variation of P/D throughput demand ratio requires phase-specific frequency control. However, evaluating the whole Azure LLM Inference Trace dataset incurs significant GPU cost and time given it only shows fluctuation on the scale of hours. Hence, we use a synthetic dataset with P/D demand ratio fluctuating in 5 minutes. Fig. 29 shows the evaluation results with LLaMA-3.1-8B using the same configuration in § VI-B.

The results shown in Tab. II confirm the adaptiveness of GreenServe. It can still maintain comparable SLO attainment rates to SGLang at maximum frequency, while obtain significant energy savings. Fig. 29 shows the frequency dynamics. When prefill throughput demand is high, the prefill instance operates at a higher frequency, while the decode instance remains at a low frequency. Conversely, when decode throughput demand is high, the decode instance operates at a higher frequency, while the prefill instance remains at a low frequency.

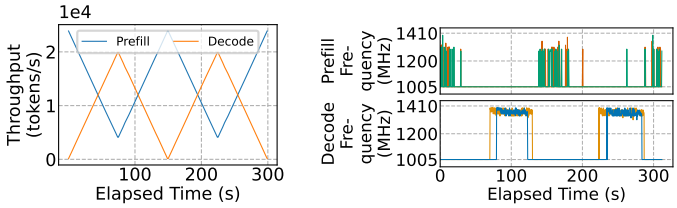


Fig. 29: Left: P/D throughput with our synthetic dataset. Right: frequency dynamics of all instances in GreenServe.

TABLE II: Latency SLO attainment and end-to-end (E2E) energy consumption under a synthetic workload with varying P/D demand ratios.

Metrics	TTFT SLO Attainment Rate (%)	ITL SLO Attainment Rate (%)	E2E Energy (J)
GreenServe	96.05	91.74	289409
SGLang-1005	77.86	36.78	257768
SGLang-1410	97.85	88.92	405340

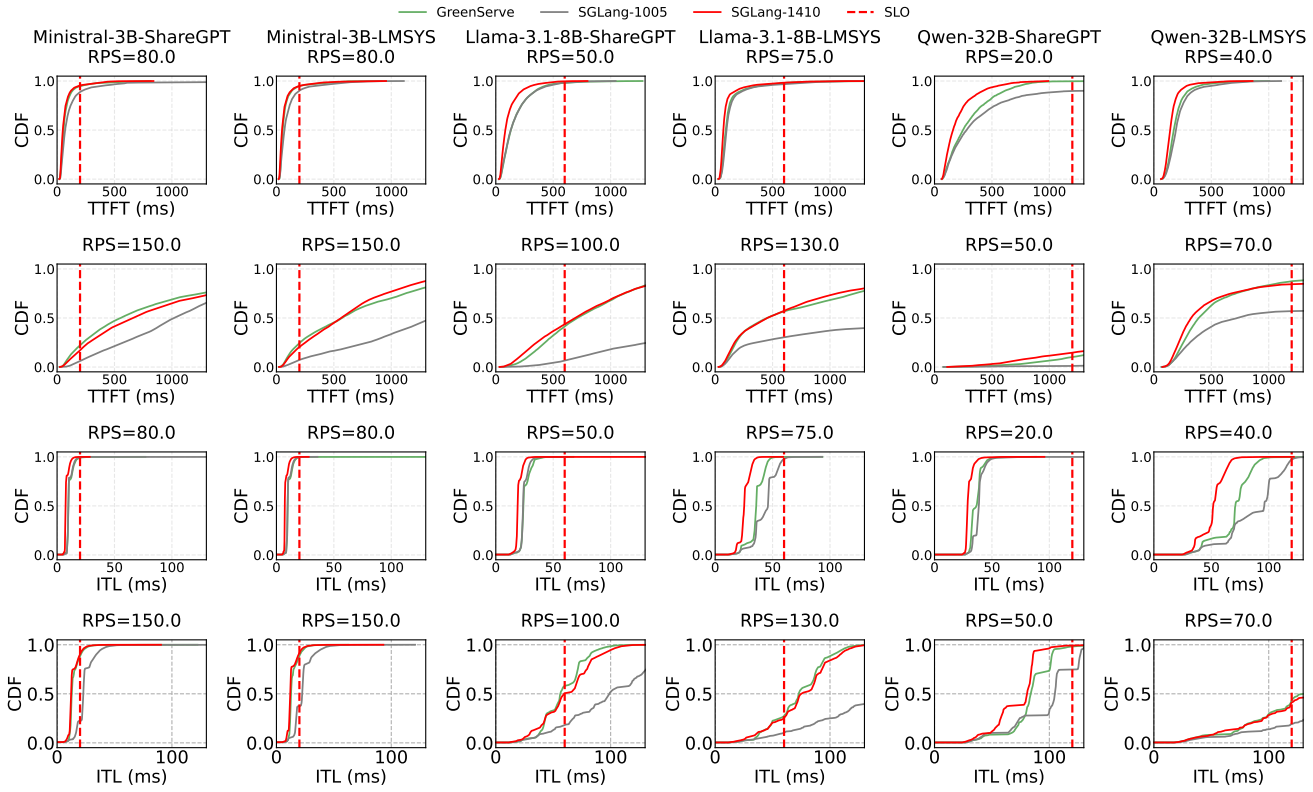


Fig. 30: CDF of TTFT (top 2 rows) and ITL (bottom 2 rows) across various models and datasets (each column). For each model and dataset combination, we pick the smallest (row 1 and 3) and the largest (row 2 and 4) request arrival rates to shows the distinct behavior of GreenServe under different workloads. Data comes from Fig. 16.