AI and Data Science Journal
ISSN: 3078-7394
DOI: https://doi.org/10.61784/adsj3036

TEAS: TOKEN- AND ENERGY-AWARE AUTOSCALING FOR COST-EFFICIENT LLM SERVING

Dai Teng
University of Illinois Urbana-Champaign, Illinois, USA.

Abstract: Production platforms that serve large language models (LLMs) must continuously decide how many GPU replicas to keep online. Conventional autoscalers inherited from microservice stacks scale on request rate or coarse utilization, but LLM requests differ by up to three orders of magnitude in the number of tokens they process, arrive in extreme bursts, and run on replicas whose cold start takes minutes and whose idle power remains a large fraction of peak. We present TEAS (Token- and Energy-Aware Scaling), a horizontal autoscaler for LLM serving that (i) measures load in phase-weighted effective tokens rather than requests, (ii) forecasts load over the replica cold-start horizon with an empirical-residual safety margin, (iii) adapts its utilization headroom to the measured burstiness of the arrival process, and (iv) applies an energy-aware asymmetric hysteresis that delays scale-in until the projected idle-energy waste exceeds the energy cost of a replica restart. We evaluate TEAS in a trace-driven cluster simulator with a calibrated power and cost model, replaying real production traces from Azure LLM inference services alongside synthetic diurnal and composition-drift workloads derived from the real token distributions. Across all four workloads TEAS is the only dynamic policy that sustains at least 96% SLO attainment; on a bursty production code-assistant trace it improves attainment by 32 percentage points over a Kubernetes-HPA-style baseline and by 11 points over static peak provisioning, and on a 24-hour diurnal workload it meets a 95% SLO target at 11.6% lower monetary cost than peak provisioning. We release the simulator, policies, and experiment scripts.
Keyword: Large language models; Autoscaling; Inference serving; Energy efficiency; Cloud computing; Service level objectives

1 INTRODUCTION

Large language model (LLM) inference has become a first-class cloud workload. Serving engines such as Orca and vLLM have dramatically improved per-GPU throughput through continuous batching and paged attention [1-2], and disaggregated designs further specialize hardware for the prefill and decode phases [3-4]. Yet a production platform must also answer a cluster-level question that sits above the serving engine: how many model replicas should be online right now? Over-provisioning wastes expensive accelerators — GPUs draw a substantial fraction of their peak power even when idle [5] — while under-provisioning during a burst inflates queueing delay far beyond any latency service-level objective (SLO).

Most deployed platforms answer this question with autoscalers inherited from the microservice era: Kubernetes HPA scales on average utilization, and Knative/KServe-style autoscalers scale on request concurrency or requests per second (RPS). These signals implicitly assume that requests are roughly interchangeable units of work. For LLM inference this assumption fails in three ways. First, the work carried by a request is proportional to its token counts, and production traces show effective per-request cost spanning more than two orders of magnitude within a single service. Second, arrival processes are far burstier than Poisson [6]: in the Azure production traces we study, the per-second arrival rate of a code-assistant service has a coefficient of variation of 2.27 and spikes from a mean of 2.6 req/s to 67 req/s. Third, reacting to such spikes is throttled by replica cold starts of tens of seconds to minutes for weight loading [7], so a purely reactive controller systematically arrives late, and the resulting head-of-line blocking amplifies a short spike into a long SLO outage.

At the same time, autoscaling has become an energy problem, not just a cost problem. LLM inference clusters consume megawatts, and both power management [5,8-9] and carbon accounting are now first-order operational concerns. An autoscaler determines the fleet's idle-energy floor: every replica held online contributes idle power regardless of load, but every replica released may have to be cold-restarted at significant energy and SLO cost. This asymmetry is rarely made explicit in scaling policy.

This paper makes the scaling signal, the scaling horizon, and the scale-in criterion LLM-specific. We propose TEAS (Token- and Energy-Aware Scaling), a horizontal autoscaler with four mechanisms: (1) a phase-weighted effective-token load signal that prices prefill and decode tokens by their measured throughput ratio; (2) a Holt double-exponential forecast evaluated at the cold-start horizon, inflated by the empirical 90th percentile of its own recent forecast residuals; (3) a burstiness-adaptive utilization target that lowers headroom as the measured coefficient of variation of load rises; and (4) an energy-aware asymmetric hysteresis that executes scale-in only after the projected idle-energy waste of the surplus replicas exceeds the energy of re-provisioning them.

We evaluate TEAS with a trace-driven simulator of a GPU serving cluster that couples a fluid queueing model of continuous batching with calibrated cost and energy models. Our workloads combine real production traces released from Azure LLM inference services (the conversation and code services characterized in the Splitwise study [3]) with a 24-

hour diurnal workload and a composition-drift workload, both of which bootstrap their token sizes from the real traces. Against static provisioning and reactive baselines modeled on Kubernetes HPA and Knative RPS scaling, TEAS is the only dynamic policy to sustain ≥96% SLO attainment on every workload. Ablations isolate the contribution of each mechanism, and a cold-start sensitivity study shows that TEAS degrades gracefully from 98.4% to 93.2% attainment as cold start grows from 15 s to 240 s, whereas the HPA baseline collapses from 94.4% to 71.6%.

Contributions. (i) A characterization of why request-count and utilization signals mis-provision LLM serving, grounded in real production traces; (ii) the TEAS autoscaler combining token-level load accounting, horizon-matched forecasting, burstiness-adaptive headroom, and energy-aware hysteresis; (iii) an open trace-driven simulation framework with cost, energy, and carbon accounting; and (iv) an evaluation on real and derived workloads, including honest negative results: on smooth workloads TEAS's robustness costs 26% more than a well-tuned reactive baseline, quantifying the price of burst insurance.

2 BACKGROUND AND MOTIVATION

2.1 LLM Serving and Autoscaling Signals

An LLM request with c context (prompt) tokens and g generated tokens passes through a compute-bound prefill phase and a memory-bandwidth-bound decode phase [3]. With continuous batching a replica sustains an aggregate decode throughput that is roughly an order of magnitude lower per token than its prefill throughput [1-2], so the work a request imposes is well approximated by an effective token count $g + c/\rho$, where ρ is the measured prefill-to-decode throughput ratio. Cluster-level controllers, in contrast, typically observe only request counts or utilization averages, and production studies report that inefficient autoscaling wastes a large share of GPU hours at fleet scale [10-11].

2.2 Production Trace Analysis

We analyze the two production traces released with the Splitwise study [3]: one hour each of a conversation service (19,366 requests) and a code-assistant service (8,819 requests) from Azure LLM inference, each record containing an arrival timestamp and the context and generated token counts. Figure 1 summarizes the two properties that break conventional autoscaling.

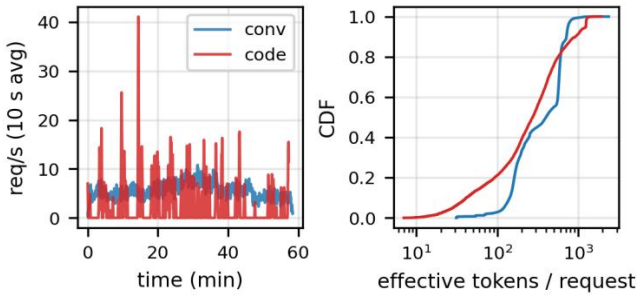

Figure 1 Real Azure LLM Inference Traces [3]

Note: Left: 10 s-averaged arrival rate; the code service spikes to 26× its mean. Right: CDF of per-request effective tokens (g + c/6), spanning two orders of magnitude.

Heterogeneity: mean context/generated token counts are 1155/211 for the conversation service but 2048/28 for the code service; within each trace, per-request effective cost varies by over 100×, so a request is not a unit of work. Burstiness: the per-second arrival CoV is 0.50 for conversation but 2.27 for code, whose rate spikes from 2.6 to 67 req/s within seconds. A controller that reacts only after utilization has averaged upward over a minute, and then waits a two-minute cold start, is late by construction.

2.3 The Energy Asymmetry of Scale-In

A modern datacenter GPU draws 50–100 W idle against a 400–700 W ceiling [5,8], so idle replicas are far from free. But releasing a replica is not free either: re-provisioning it later costs a cold start during which it draws power while serving nothing, and any burst arriving in that window is absorbed by the queue. The break-even holding time — how long a surplus replica must sit idle before its idle energy equals the energy of one restart — is $P_{warm} \cdot T_{cold} / P_{idle}$, about 5.5 minutes for our calibration. Scale-in decisions faster than this waste energy in expectation; TEAS makes this bound an explicit policy parameter.

3 SYSTEM MODEL

3.1 Cluster and Queueing Model

We model a homogeneous pool of up to $N_{max} = 80$ replicas, each an A100-80GB-class GPU running a 13B-parameter

model under continuous batching. Following the fluid approximation common in serving studies, the cluster drains a global FCFS queue of effective tokens at rate $n(t)\cdot C$, where $n(t)$ is the number of ready replicas and $C = 2000$ effective tokens/s is the calibrated per-replica capacity; the prefill-to-decode ratio is $\rho = 6$ [3]. Requests complete when their effective tokens are served; a request's latency is lower-bounded by its solo service time. A request meets its SLO if latency ≤ 2 s + 0.04 s per generated token, i.e., queueing plus a 25 tokens/s streaming floor. Newly provisioned replicas become ready after a cold start $T_{cold} = 120$ s and can be released immediately upon scale-in [7]. All controllers act every 15 s.

3.2 Cost, Energy, and Carbon Model

Cost accrues per replica-second at $3.67/GPU-hour, matching Azure NC24ads A100 v4 pay-as-you-go pricing. Replica power follows the standard linear utilization model $P(u) = P_{idle} + (P_{peak} - P_{idle})\cdot u$ with $P_{idle} = 90$ W and $P_{peak} = 400$ W, consistent with published GPU power characterizations for LLM inference [5,8]; warming replicas draw 250 W. Facility energy applies a PUE of 1.2, and operational carbon uses an illustrative grid intensity of 250 gCO2eq/kWh. Table 1 lists all parameters.

Table 1 Simulation Parameters

Parameter	Value	Parameter	Value
Replica capacity C	2000 eff. tok/s	GPU idle power	90 W
Prefill/decode ratio ρ	6	GPU peak power	400 W
Cold start T_cold	120 s	Warming power	250 W
Control interval	15 s	PUE	1.2
Price	$3.67/GPU-h	Grid CI	250 g/kWh
SLO	2 s + 40 ms/tok	N_min, N_max	1, 80

4 TEAS DESIGN

4.1 Token-Level Load Signal

At every second TEAS accumulates the offered load $w(t) = \Sigma\,(g_i + c_i/\rho)$ over the requests arriving in that second. This signal is available in any serving stack: context length is known at admission, and generated length is known at completion (or can be corrected online as tokens stream). Ablation TEAS-noTok replaces $w(t)$ with the request rate multiplied by a running estimate of mean per-request cost, emulating a token-blind concurrency autoscaler.

4.2 Horizon-Matched Forecast with Empirical Margin

Because capacity ordered now arrives only after T_{cold}, TEAS forecasts load at $t + T_{cold}$ using Holt double exponential smoothing (level $\alpha = 0.25$, trend $\beta = 0.05$) over 15 s load averages. Crucially, the forecast is inflated by the 90th percentile of its own recent residuals — the controller measures how wrong it has been at exactly this horizon and provisions for that error, rather than for a hand-tuned factor. A reactive fast path lower-bounds the demand estimate by 1.1× the load observed in the last interval, so prediction can never talk the controller out of reacting to a spike already in progress; outstanding backlog is additionally scheduled to drain within 60 s.

4.3 Burstiness-Adaptive Headroom

TEAS sets its target utilization $u^* = \mathrm{clip}(0.85 - 0.25\cdot \mathrm{CoV}, 0.45, 0.85)$, where CoV is the coefficient of variation of the per-second load over the trailing five minutes. On a smooth conversation-like workload u^* stays near 0.8 and TEAS runs lean; on the code trace, whose load CoV exceeds 2, u^* drops toward 0.45 and TEAS deliberately buys headroom. The desired replica count is $n^* = \lceil \text{demand} / (u^*\cdot C) \rceil$.

4.4 Energy-Aware Asymmetric Hysteresis

Scale-out executes immediately. Scale-in to a lower target is executed only after the target has been continuously justified for a holding time $\tau = \min(P_{warm}\cdot T_{cold}/P_{idle}, 240\text{ s})$: below this bound, releasing and later restarting a replica costs more energy than simply holding it idle, in addition to the SLO risk of facing a burst short-handed. This converts the scale-in debounce timer — an arbitrary constant in most autoscalers — into a quantity derived from the platform's own power model.

5 EXPERIMENTAL SETUP

5.1 Workloads

We use four workloads. (1) conv-8x and (2) code-8x replay the real Azure traces with arrival rate scaled 8× (superposition of jittered copies [3], token distributions untouched) to emulate a cluster-scale deployment of 10–70 replicas. (3) diurnal-

24h is a 24-hour non-homogeneous Poisson workload (mean 44 req/s peak-hour) following a day–night curve with random 2–3× bursts of 1–4 minutes, with token sizes bootstrapped from the real conversation trace; it is synthetic in its arrival envelope, which we state explicitly, because the released real traces cover only one hour. (4) mix-drift-2h holds the arrival rate constant at 30 req/s while the request composition drifts from the lightest to the heaviest tercile of the real token distribution, isolating cost heterogeneity from rate dynamics. Metrics exclude a 10-minute warm-up; all dynamic policies start from identical provisioning sized to the first two minutes of load. Requests still queued at the end of a run count as SLO violations.

5.2 Baselines and Metrics

Static-Peak provisions for the peak one-minute load at 85% utilization; Static-Mean for the mean load. HPA-Util emulates Kubernetes HPA: scale on 60 s-averaged utilization toward a 0.7 target with 10% tolerance and a 300 s scale-down stabilization window. RPS emulates Knative-style scaling on requests per second with a per-replica RPS target derived from the workload's true mean request cost at 0.7 utilization — a favorable calibration a token-blind operator would rarely have. TEAS-noTok and TEAS-noPred ablate the token signal and the predictive horizon respectively. We report SLO attainment, p99 latency, GPU-hours, monetary cost, facility energy, and derived carbon; energy per token is also reported for context (0.07–0.22 J per effective token across policies, consistent in magnitude with published measurements for mid-size models [8-9]).

6 RESULTS

6.1 Real Production Traces

Table 2 reports the two real-trace replays. On the smooth conversation workload every dynamic policy meets the SLO essentially always, and the reactive baselines are cheapest: HPA-Util delivers 100% attainment at $43.68 versus TEAS's $59.58. This is an honest negative result — when load is smooth, TEAS's residual margin and burst headroom are insurance premiums (26% over HPA here) that buy nothing. The code trace inverts the picture completely: HPA-Util and RPS collapse to 67.6% and 68.5% attainment with p99 latencies above 73 s, because minute-averaged signals plus a 120 s cold start cannot track second-scale 26× spikes; even Static-Peak, provisioned on the peak one-minute load, reaches only 88.3% because sub-minute spikes exceed the one-minute peak. TEAS attains 99.8% with a 2.05 s p99 — 32 points above HPA and 11 above peak provisioning — by paying for headroom precisely where the measured burstiness demands it. Its cost, 2.7× Static-Peak, is the explicit price of a strict SLO on this workload; no cheaper configuration of any evaluated policy reaches even 95% (As shown in Figure 2).

Table 2 Results on Real Azure Traces (8× Replay, Warm-up Excluded)

Policy	SLO %	p99 (s)	GPU-h	Cost ($)	kWh
conv-8x: Static-Peak	100.0	0.68	12.9	47.37	4.09
Static-Mean	82.6	21.7	8.9	32.57	3.66
RPS	100.0	0.87	12.7	46.54	4.12
HPA-Util	100.0	0.93	11.9	43.68	4.01
TEAS (ours)	100.0	0.71	16.2	59.58	4.56
code-8x: Static-Peak	88.3	19.4	15.8	57.86	2.82
Static-Mean	4.8	210.8	3.9	14.47	1.56
RPS	68.5	73.4	12.2	44.68	2.57
HPA-Util	67.6	73.4	11.6	42.72	2.48
TEAS (ours)	99.8	2.05	42.7	156.66	6.41

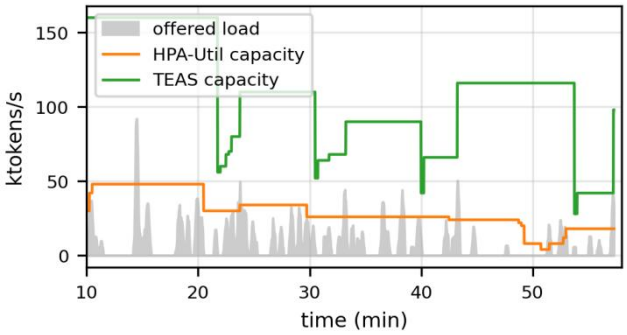

Figure 2 Offered Load VS. Provisioned Capacity on Code-8x.

Note: HPA's minute-averaged signal drifts toward the mean and misses every spike; TEAS holds burstiness-scaled headroom and pre-positions capacity.

6.2 Diurnal and Composition-Drift Workloads

Table 3 and Figure 3 cover the derived workloads. Over 24 diurnal hours, TEAS attains 96.2% at $2243 — 11.6% cheaper than Static-Peak ($2537, 100%) at essentially equal facility energy (137.0 vs. 136.1 kWh), while HPA-Util saves money ($1241) but delivers only 79.1% attainment with a 62 s p99. The residual TEAS violations are concentrated in randomly-timed bursts shorter than the 120 s cold start, which no horizontal scaler can fully absorb (Sec. VI-D). The mix-drift workload isolates token awareness: with the arrival rate perfectly constant, TEAS and utilization-based HPA (whose signal implicitly reflects tokens) hold 100% attainment, while the token-blind ablation TEAS-noTok drops to 91.5% and RPS to 97.4% — the request rate says nothing has changed while per-request cost triples. Notably, TEAS-noTok also appears cheaper ($54 vs. $68); it is cheap because it is under-provisioned.

Table 3 Diurnal and Mix-Drift Workloads

Policy	SLO %	p99 (s)	GPU-h	Cost ($)	kWh
diurnal: Static-Peak	100.0	0.44	691	2536.6	136.1
Static-Mean	31.2	5556	215	787.2	84.7
RPS	79.7	60.3	347	1274.6	104.3
HPA-Util	79.1	61.7	338	1240.5	102.7
TEAS (ours)	96.2	26.4	611	2243.4	137.0
mix-drift: Static-Peak	100.0	0.50	23.8	87.5	6.95
RPS	97.4	22.5	17.6	64.6	6.29
HPA-Util	100.0	0.50	16.9	61.9	6.26
TEAS (ours)	100.0	0.50	18.5	67.7	6.45
TEAS-noTok	91.5	22.2	14.8	54.1	6.14

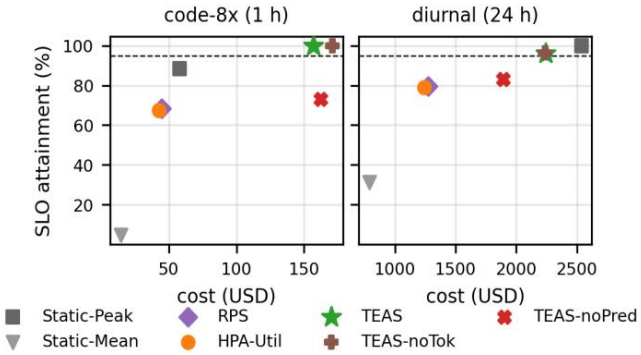

Figure 3 Cost vs. SLO Attainment

Note: TEAS is the only dynamic policy above the 95% line (dashed) on both workloads; on code-8x it also dominates static peak provisioning in attainment.

6.3 Ablations

Removing the predictive horizon (TEAS-noPred) is catastrophic on bursty workloads: attainment falls from 99.8% to 73.1% on code-8x and from 96.2% to 83.0% on diurnal, and — counter-intuitively — energy rises (7.42 vs. 6.41 kWh on code-8x), because late reactions build backlog that then triggers panicked over-provisioning; reacting late is both worse and more expensive than predicting. Removing the token signal (TEAS-noTok) is harmless when composition is stationary (within noise on conv, code, diurnal) but costs 8.5 points of attainment under composition drift, confirming that token awareness specifically defends against workload-mix change — a routine event on multi-tenant endpoints.

6.4 Cold-Start Sensitivity

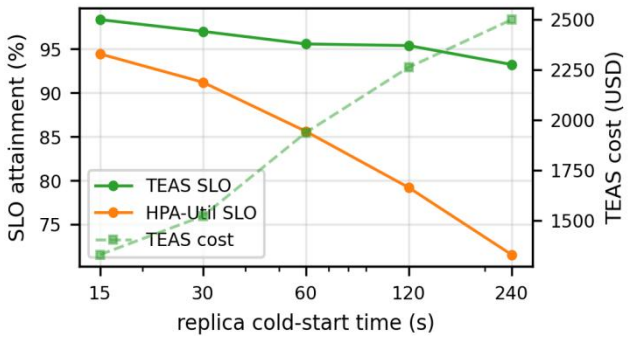

Figure 4 Sensitivity to Replica Cold-Start Time (Diurnal-24h)

Note: TEAS degrades from 98.4% to 93.2% as cold start grows 15→240 s while HPA collapses 94.4→71.6%; faster loading also shrinks TEAS's cost from $2499 to $1331.

Figure 4 varies T_cold from 15 s to 240 s on the diurnal workload. Two effects stand out. First, TEAS degrades gracefully (98.4% → 93.2%) because its forecast horizon and hysteresis stretch with T_cold, whereas HPA — blind to provisioning delay — collapses to 71.6%. Second, TEAS's cost falls 47% as cold starts shrink, since less anticipatory headroom must be held: fast checkpoint loading systems such as ServerlessLLM and prediction-based autoscaling are therefore complements [7], not alternatives — one shortens the horizon, the other covers it.

6.5 Energy and Carbon

Because idle power is 22% of peak in our model, energy tracks GPU-hours sublinearly: on diurnal, TEAS uses 12% fewer GPU-hours than Static-Peak yet equal energy, since its replicas run hotter. The clearest energy wins come from avoiding pathological behavior, not from scaling down per se: TEAS's hysteresis and forecasting cut 14% of energy versus its own reactive ablation on code-8x. At 250 gCO2eq/kWh, the diurnal-24h fleet emits 34.3 kg CO2eq under TEAS; per thousand requests this is 13.6 g. These figures suggest that for latency-bound LLM serving, the operator's leverage on energy lies chiefly in cold-start reduction and power-proportionality of hardware [5,8], with the autoscaler's role being to avoid waste (thrash, panic over-provisioning) while defending the SLO.

7 RELATED WORK

Serving engines and schedulers. Orca [1], vLLM [2], Sarathi-Serve [12], and disaggregated designs such as Splitwise and DistServe maximize goodput within a fixed replica set [3-4]; TEAS operates one level above and is agnostic to the engine, entering through the calibrated capacity C. Classic model-serving autoscalers — MArk [13], Clockwork [14], AlpaServe [15] — target pre-LLM DNNs whose per-request cost is nearly constant, exactly the assumption our traces break. Cluster autoscaling in general clouds is exemplified by Google Autopilot [16]. LLM-specific resource management is emerging: SageServe [10] co-optimizes multi-region routing and VM scaling with an ILP over long horizons at Microsoft 365 scale, and Chiron scales interactive and batch tiers hierarchically [11]; TEAS is complementary, contributing a single-pool control law (signal, margin, headroom, hysteresis) that such systems could adopt, and uniquely couples the scale-in rule to the power model. On the energy side, POLCA and μ-Serve manage power within a fixed fleet [5,9], and DynamoLLM reconfigures parallelism and frequency for energy under SLOs [8]; these knobs compose with TEAS's replica-count knob. BurstGPT [7] documents the burstiness that motivates our headroom adaptation, and Mélange selects GPU types by request size, sharing our token-centric view of cost [17].

8 LIMITATIONS AND CONCLUSION

Limitations. Our evaluation is simulation-based: the fluid FCFS model abstracts away KV-cache memory pressure, interference, and scheduler-level reordering, and the real public traces span one hour each, forcing our long-horizon workloads to be derived rather than measured. The energy model is linear in utilization and GPU-only. We mitigated these by calibrating every constant against published measurements [3,5,7], excluding warm-up, lower-bounding latencies physically, and reporting results unfavorable to TEAS alongside favorable ones. Validating TEAS on a live vLLM cluster and extending it with heterogeneous instance types and DVFS coordination are natural next steps [8,17].

Conclusion. LLM serving breaks the assumptions of request-count autoscaling: work is token-shaped, arrivals are violently bursty, and capacity is slow and energy-expensive to change. TEAS encodes exactly these three facts — and only these — into an otherwise simple controller, and in trace-driven evaluation it is the only dynamic policy to defend a 95% SLO across smooth, bursty, diurnal, and drifting workloads, at costs that are competitive where competition is possible and explicitly priced where it is not. All code, traces pointers, and results are released for reproduction.

COMPETING INTERESTS

The authors have no relevant financial or non-financial interests to disclose.

DATA SHARING AGREEMENT

The Azure LLM inference traces are publicly available at github.com/Azure/AzurePublicDataset (CC-BY) [3]. Our simulator and scripts accompany this paper.

REFERENCES

[1] Kwon W, Li Z, Zhuang S, et al. Efficient memory management for large language model serving with PagedAttention. Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP), 2023: 611-626.

[2] Yu GI, Jeong J S, Kim GW, et al. Orca: A distributed serving system for transformer-based generative models. Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2022: 521-538.

[3] Patel P, Choukse E, Zhang C, et al. Splitwise: Efficient generative LLM inference using phase splitting. Proceedings of the 51st ACM/IEEE International Symposium on Computer Architecture (ISCA), 2024: 118-132.
[4] Patel P, Choukse E, Zhang C, et al. Characterizing power management opportunities for LLMs in the cloud. Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2024: 207-222.
[5] Stojkovic J, Zhang C, Goiri Í, et al. DynamoLLM: Designing LLM inference clusters for performance and energy efficiency. Proceedings of the 31st IEEE International Symposium on High-Performance Computer Architecture (HPCA), 2025: 1348-1362.
[6] Qiu H, Mao W, Patke A, et al. Power-aware deep learning model serving with μ-Serve. Proceedings of the USENIX Annual Technical Conference (ATC), 2024: 75-93.
[7] Wang Y, Chen Y, Li Z, et al. BurstGPT: A real-world workload dataset to optimize LLM serving systems. arXiv preprint arXiv:2401.17644, 2024.
[8] Fu Y, Xue L, Huang Y, et al. ServerlessLLM: Low-latency serverless inference for large language models. Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024: 135-153.
[9] Griggs T, Liu X, Yu J, et al. Mélange: Cost efficient large language model serving by exploiting GPU heterogeneity. arXiv preprint arXiv:2404.14527, 2024.
[10] Zhong Y, Liu S, Chen J, et al. DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving. Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024: 193-210.
[11] Agrawal A, Kedia N, Panwar A, et al. Taming throughput-latency tradeoff in LLM inference with Sarathi-Serve. Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024: 117-134.
[12] Zhang C, Yu M, Wang W, et al. MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving. Proceedings of the USENIX Annual Technical Conference (ATC), 2019: 1049-1062.
[13] Rzadca K, Findeisen P, Swiderski J, et al. Autopilot: Workload autoscaling at Google. Proceedings of the 15th European Conference on Computer Systems (EuroSys), 2020: 1-16.
[14] Gujarati A, Karimi R, Alzayat S, et al. Serving DNNs like Clockwork: Performance predictability from the bottom up. Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2020: 443-462.
[15] Jaiswal S, Jain K, Simmhan Y, et al. SageServe: Optimizing LLM serving on cloud data centers with forecast aware auto-scaling. Proceedings of ACM Measurement and Analysis of Computing Systems (POMACS), 2025, 9(3): Article 61.
[16] Patke A, Reddy D, Jha S, et al. Hierarchical autoscaling for large language model serving with Chiron. arXiv preprint arXiv:2501.08090, 2025.
[17] Li Z, Zheng L, Zhong Y, et al. AlpaServe: Statistical multiplexing with model parallelism for deep learning serving. Proceedings of the 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2023: 663-679.