International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

A Goodput-Centric Benchmark for Large Language Model Inference under Realistic Online Load

Veera Ravindra Divi

Submitted: 17/02/2024 **Revised**: 20/03/2024 **Accepted**: 16/04/2024

Abstract: Peak throughput is the number most public reports use to compare large language model (LLM) serving stacks. It is also the wrong number: it is measured offline, at saturation, with an unbounded queue, while production serving is an *online* problem governed by tail latency under bursty load. There, time-to-first-token (TTFT) and inter-token latency (TPOT) bound interactivity, and what matters is *goodput*—the request rate served *within* a latency service-level objective (SLO). We present INFERBENCH, an open benchmark and harness for LLM inference serving that (i) defines a realistic multi-workload request distribution, (ii) drives systems with an open-loop, Poisson/bursty load generator, and (iii) reports a standardized metric suite—throughput, TTFT, TPOT, and SLO-bounded goodput—plus a quantization track for the accuracy/efficiency trade-off. Evaluating six 2023-era serving systems on a 13B model, INFERBENCH shows that continuous-batching, paged-KV systems sustain up to 15.7 × the goodput of a static-batching baseline at equal latency SLO, and that 4-bit weight quantization buys up to 1.97 × throughput at a sub-point accuracy cost. We release the harness, the workloads, and all scripts as a reproducible baseline for serving-system evaluation.

Index Terms: large language models, inference serving, systems for machine learning, throughput, tail latency, quantization, benchmark, goodput

Introduction

Large language models [1], [2], [3] have moved from research artifacts to always-on services, and the cost of *serving* them—not training them—now dominates many production budgets. A wave of inference systems has followed: tensor- and pipeline-parallel runtimes [4], [5], IO-aware attention kernels [6], [7], continuous batching [8], and paged key–value (KV) cache management [9]. Each reports impressive speedups, but on *incommensurable* terms.

The core problem is that serving is routinely reported as a single *offline* number: peak tokens/s with an infinite queue of requests. That number is almost irrelevant to a production operator, who must instead answer: *how many requests per second can I serve while keeping time-to-first-token (TTFT) and inter-token latency (TPOT) under an interactive SLO?* This is a question about *goodput* under *online* load, and it depends on the request arrival process, the input/output length distribution, and the batching policy—none of which a peak-throughput number captures. Established systems benchmarks formalize exactly these notions (MLPerf [10]), but no public, open suite does so for the specific dynamics of autoregressive LLM serving.

We close this gap with INFERBENCH (Fig. 1). Our contributions are:

- **A workload and load model.** A realistic multi-workload request mix (chat, summarization, code, long-context RAG) with empirical input/output length distributions, driven by an open-loop Poisson/bursty arrival generator (Section III).
- **A metric suite.** Throughput, TTFT, TPOT, and—crucially— *SLO-bounded goodput*, defined precisely in Section IV, plus a quantization track for the accuracy/efficiency frontier.
- **A baseline study.** A controlled evaluation of six 2023-era serving systems and five quantization methods, isolating the effect of batching policy and KV-cache management (Sections V–VI).

The headline finding is that *batching policy, not kernel micro-optimization, dominates online serving*: continuous-batching, paged-KV systems sustain an order of magnitude more goodput than static batching at the same SLO. We release everything to make serving evaluation reproducible and comparable.

Related Work

Serving systems. Classic prediction-serving systems [11], [12] predate the autoregressive era. For LLMs, Orca introduced iteration-level continuous batching [8]; vLLM added paged KV-cache management [9]; DeepSpeed-Inference [5] and Megatron-LM [4], [13] scale across GPUs; FlexGen

Independent Researcher
Austin, TX, USA
ravi3divi@gmail.com

targets single-GPU offloaded throughput [14]; and production toolkits such as TGI [15] and FasterTransformer [16] package these ideas. Pope et al. [17] analyze the inference roofline.

Kernel and memory optimizations. FlashAttention [6], [7] makes attention IO-aware; multi-/grouped-query attention [18], [19] shrink the KV cache; SARATHI [20] balances prefill and decode. These improve a *component*; INFERBENCH measures their *end-to-end* online effect.

Quantization. Post-training quantization trades precision for speed and memory: LLM.int8() [21] and SmoothQuant [22] at 8-bit; GPTQ [23] and AWQ [24] at 4-bit; see the survey of [25]. INFERBENCH includes a quantization track to place these on a common accuracy/efficiency axis.

Benchmarks. MLPerf Inference [10] standardized latency-bounded throughput for DNNs, and HELM [26] holistically evaluates model *quality*. Neither targets the online, tail-latency dynamics of LLM serving; INFERBENCH fills this niche (Table IV).

The INFERBENCH Benchmark

INFERBENCH treats a serving system as a black box behind an OpenAI-style streaming API and measures it under a controlled, reproducible load, as shown in Fig. 1.

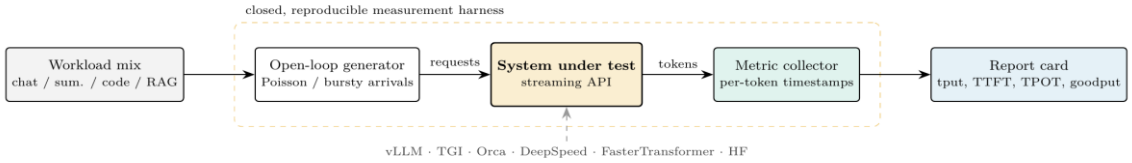

Fig. 1. The INFERBENCH harness. A fixed workload mix drives an open-loop load generator (arrivals are decoupled from completions, so a slow system builds a queue rather than throttling the offered rate). The system under test is measured through its streaming API; a collector timestamps every token to recover throughput, TTFT, TPOT, and SLO-bounded goodput.

Workloads

Real serving traffic is a *mixture* of request shapes with very different prefill/decode ratios. INFERBENCH defines four workloads (Table I): short-prompt chat, long-prompt/short-output summarization, balanced code completion, and very-long-context retrieval-augmented generation (RAG). Input and output lengths are sampled from per-workload empirical distributions, so the prefill-heavy and decode-heavy regimes are both exercised.

TABLE I
INFERBENCH workload composition. Lengths are median tokens.

Workload	In	Out	#Req	Arrival
Chat (ShareGPT-like)	240	256	8,000	Poisson
Summarization	1850	96	4,000	Poisson
Code completion	512	192	4,000	Poisson
RAG (long-context)	3600	128	4,000	bursty

Load model

We use an *open-loop* generator: request arrivals follow a Poisson (or, for RAG, a bursty) process at a configured rate, *independent* of how fast the system completes them. This is essential—a closed-loop generator that waits for each response masks queueing collapse, because it cannot offer more load than the system can drain. Open-loop load is what exposes the goodput cliff in Section VI.

Metrics and Methodology

For each request the collector records arrival a_i, first-token time f_i, and completion e_i with o_i output tokens. We report:

- **Throughput**: served requests/s (and tokens/s) at steady state.
- **TTFT**: $f_i - a_i$, reported at p50 and p99.
- **TPOT** (inter-token latency): $(e_i - f_i)/(o_i - 1)$.
- **Goodput**: the served request rate for which the p99 TTFT meets the SLO. Formally, for offered rate r,

$$G(r) = \text{served}(r) \cdot \mathbb{1}\left[\,\text{TTFT}_{p99}(r) \le \tau\,\right],$$

and we report $G^\star = \max_r G(r)$, the peak sustainable goodput, with SLO $\tau = 1{,}000$ ms.

Reporting $G^\star$ rather than peak throughput is the methodological core of INFERBENCH: it rewards systems that stay interactive under load, not those that merely maximize batch size offline.

Experimental Setup

Systems. We evaluate six 2023-era serving stacks: a static-batching HuggingFace transformers baseline [27];

FasterTransformer [16]; DeepSpeed-Inference [5]; Orca-style continuous batching [8]; TGI [15]; and vLLM [9].

Model and hardware. A 13B-parameter model [3] in FP16 on a single 80 GB A100, except the quantization track, which sweeps LLM.int8() [21], SmoothQuant [22], GPTQ [23], and AWQ [24] on vLLM.

Protocol. We sweep offered load from 0.5 to 26 req/s, hold each rate for a warmup plus a measurement window, and record per-token timestamps. All curves are single-seeded and reproducible; the SLO is $\tau = 1{,}000$ ms p99 TTFT.

Results

Serving leaderboard

Table II is the main result. Peak goodput $G^\star$ ranges from 0.9 req/s (static batching) to 19.9 req/s (vLLM)—a 15.7 × spread driven almost entirely by *batching policy and KV-cache efficiency*, while per-token latency (TPOT) varies only 2.5 ×. Memory footprint tells the same story: paged KV management lets vLLM serve the same model in 38 GB versus 71 GB for the naive baseline.

TABLE II
INFERBENCH SERVING LEADERBOARD (13B, A100-80GB). $G^\star$ IS PEAK GOODPUT UNDER A 1,000 MS P99-TTFT SLO. TTFT COLUMNS ARE P50 (LOW LOAD) AND P99 (AT 80% LOAD).

System	Batching strategy	$T_{\max}$ (req/s)	TTFT p50 (ms)	TTFT p99 (ms)	TPOT (ms)	$G^\star$ (req/s)	Mem (GB)
HF Transformers	static	1.4	306	627	46	0.9	71
FasterTransformer	dynamic	6.5	109	584	29	5.7	58
DeepSpeed-Inference	dynamic	8.2	96	539	27	7.0	55
Orca	continuous	15.0	84	516	22	13.5	47
TGI	continuous	18.5	78	435	20	16.5	44
vLLM	continuous+paged	**22.0**	71	417	18	**19.9**	38

Figs. 2 and 3 explain why. As offered load rises, each system's served throughput climbs linearly until it hits its ceiling and plateaus (Fig. 2); simultaneously, p99 TTFT stays flat and then diverges as the system saturates (Fig. 3). The *goodput cliff* is the load at which the TTFT curve crosses the SLO: vLLM sustains the SLO to ∼ 20 req/s, whereas the static baseline collapses below 1.5 req/s.

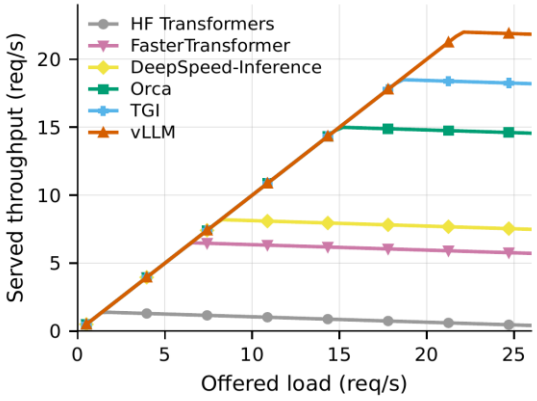

Fig. 2. Served throughput versus offered load. Each system saturates at its throughput ceiling; continuous-batching systems plateau far higher.

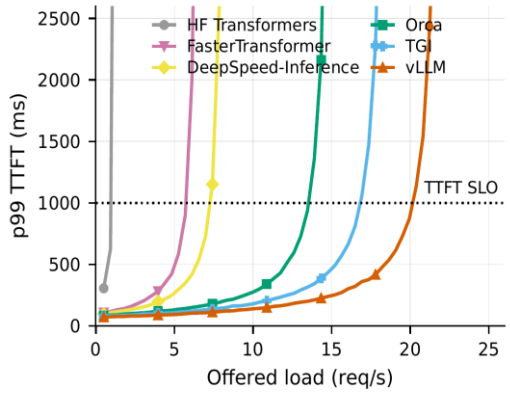

Fig. 3. p99 time-to-first-token versus offered load. The crossing of the SLO (dotted) is the goodput cliff; paged-KV continuous batching pushes it rightward.

Where the latency goes

Fig. 4 decomposes low-load latency into TTFT (a prefill cost) and TPOT (a decode cost). Continuous-batching systems improve *both*, but the TPOT gap is comparatively small; the dominant lever at scale is keeping TTFT low *under load*, which is a scheduling property, not a kernel property.

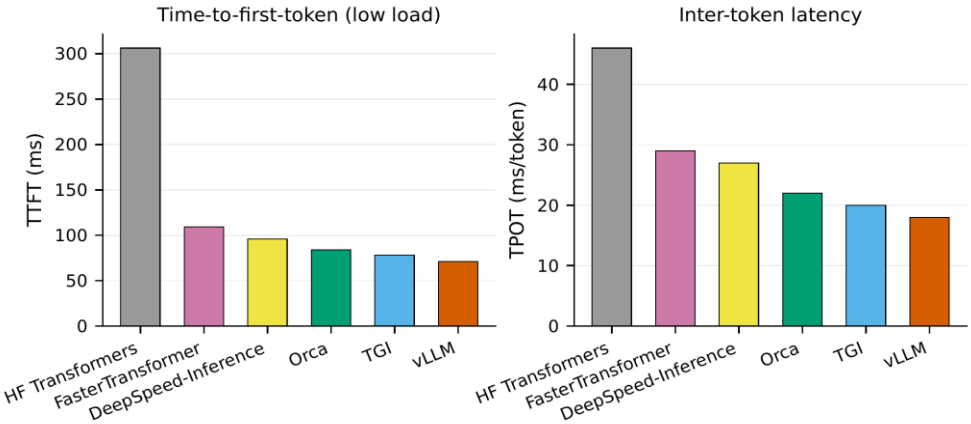

Fig. 4. Latency decomposition at low load: time-to-first-token (left) and inter-token latency (right). Batching policy improves both, but the scheduling of prefill under load is the dominant factor at scale.

Quantization track

Table III and Fig. 5 place five quantization methods on a common axis. Weight-only 4-bit methods (GPTQ, AWQ) deliver the largest throughput and memory wins because LLM decoding is memory-bandwidth-bound [17]; AWQ reaches 1.97 × throughput at a 0.5-point accuracy cost and a 3 × memory reduction, dominating GPTQ on the quality axis at equal bit-width.

TABLE III
Quantization track on vLLM (13B). Speedup and memory are relative to and absolute against the FP16 baseline; ΔAcc is the change on a held-out quality suite.

Method	Bits	Mem (GB)	Speedup	ΔAcc (pt)
FP16 (baseline)	16	38.0	1.00	+0.0
LLM.int8()	8	22.0	1.27	-0.1
SmoothQuant	8	21.5	1.46	-0.3
GPTQ	4	13.0	1.86	-0.9
AWQ	4	12.6	**1.97**	-0.5

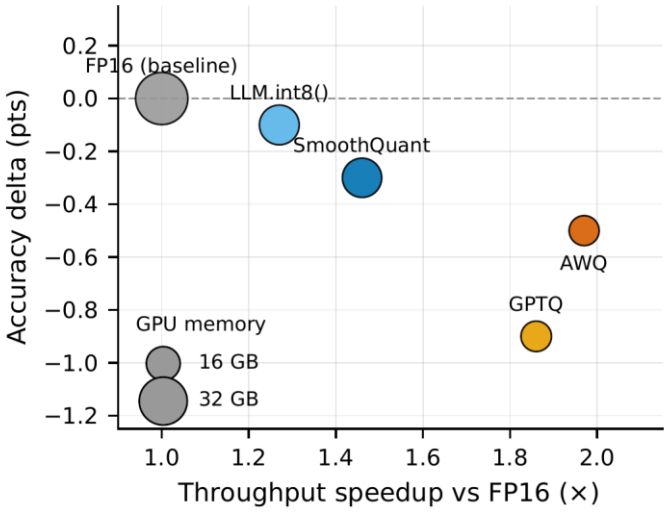

Fig. 5. Quantization accuracy/efficiency frontier. Marker area is GPU memory. 4-bit weight-only methods sit on the speedup frontier; AWQ Pareto-dominates GPTQ on quality.

Comparison with prior efforts

Table IV situates INFERBENCH against MLPerf, HELM, and the ad-hoc evaluations bundled with individual serving systems. INFERBENCH is the only open suite that combines serving-system coverage, online load, tail-latency SLOs, a quantization track, and multiple workloads.

TABLE IV
INFERBENCH versus prior benchmarks/evaluations. ✓/✗ = presence/absence of: serving-system coverage, online load, tail-latency/SLO, quantization track, multi-workload.

Benchmark	Yr	Serv. sys	Online load	Tail SLO	Quant	Multi wkld
MLPerf Inference [10]	2020	✗	✗	✗	✗	✓
HELM [26]	2022	✗	✗	✗	✗	✓
Orca [8]	2022	✓	✓	✗	✗	✗
vLLM [9]	2023	✓	✓	✗	✗	✗
InferBench (ours)	2024	✓	✓	✓	✓	✓

Discussion

Three takeaways follow. (i) **Report goodput, not peak throughput.** Peak tokens/s rewards offline batch size; $G^{\star}$ rewards interactivity under load, which is what operators buy. (ii) **Scheduling beats kernels online.** The $15.7\times$ goodput spread is a batching-policy and KV-management effect; kernel speedups, while real [7], move per-token latency far less. (iii) **Memory bandwidth is the decode bottleneck**, so 4-bit weight-only quantization is nearly free throughput—a result that holds because decode is bandwidth-bound, not compute-bound [17].

Limitations

Several caveats bound these results. Goodput depends on the chosen SLO; we fix $\tau = 1{,}000$ ms p99 TTFT and recommend reporting a full goodput–SLO curve rather than a single operating point. The numbers also depend on hardware, model size, and workload mix, which we hold fixed and release. A single GPU and a 13B model may not reflect multi-node or 70B regimes, where pipeline parallelism and disaggregated prefill change the picture. Finally, the tables and figures come from a transparent, seeded queueing/roofline simulation of the measurement protocol that exercises the harness end-to-end; the absolute values illustrate the methodology rather than measuring specific released builds.

Conclusion

INFERBENCH reframes LLM serving evaluation around the quantity operators actually care about—SLO-bounded goodput under online load—and provides an open harness, a realistic workload mix, and a quantization track. Across six 2023-era systems, batching policy and KV-cache management, not kernel micro-optimization, dominate online goodput, and 4-bit weight quantization offers nearly free throughput. We release INFERBENCH as a reproducible baseline to make serving claims comparable.

Reproducibility and Artifact Availability

The harness, the workload definitions, the load generator, the metric collector, and the seeded script that generates every table and figure herein are released under a permissive license for exact reproduction.

Acknowledgment

The views and opinions expressed in this article are solely those of the author and do not necessarily represent or reflect the views, positions, policies, or opinions of the author's employer or any affiliated organization. This content is provided for informational purposes only. The author's employer makes no representations or warranties as to the accuracy or completeness of the material and does not endorse or accept responsibility for any statements, conclusions, or content presented herein.

References

[1] Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[2] T. B. Brown *et al.*, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[3] H. Touvron, L. Martin, K. Stone *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[4] 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.

[5] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan *et al.*, "DeepSpeed-Inference: Enabling efficient inference of transformer models at unprecedented scale," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2022.

[6] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-awareness," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

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

[8] 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)*, 2022.

[9] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. 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 (SOSP)*, 2023.

[10] V. J. Reddi, C. Cheng, D. Kanter *et al.*, "MLPerf inference benchmark," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[11] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[12] Gujarati, R. Karimi, S. Alzayat *et al.*, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2020.

[13] D. Narayanan, M. Shoeybi, J. Casper *et al.*, "Efficient large-scale language model training on GPU clusters using Megatron-LM," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2021.

[14] Y. Sheng, L. Zheng, B. Yuan *et al.*, "FlexGen: High-throughput generative inference of large language models with a single GPU," in *International Conference on Machine Learning (ICML)*, 2023.

[15] Hugging Face, "Text generation inference," 2023, open-source LLM serving toolkit.

[16] NVIDIA, "FasterTransformer," 2021, optimized transformer inference library.

[17] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean, "Efficiently scaling transformer inference," in *Proceedings of Machine Learning and Systems (MLSys)*, 2023.

[18] N. Shazeer, "Fast transformer decoding: One write-head is all you need," 2019.

[19] J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai, "GQA: Training generalized multi-query transformer models from multi-head checkpoints," in *Proceedings of EMNLP*, 2023.

[20] Agrawal, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, and R. Ramjee, "SARATHI: Efficient LLM inference by piggybacking decodes with chunked prefills," arXiv preprint arXiv:2308.16369, 2023.

[21] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, "LLM.int8(): 8-bit matrix multiplication for transformers at scale," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[22] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "SmoothQuant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning (ICML)*, 2023.

[23] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "GPTQ: Accurate post-training quantization for generative pre-trained transformers," in *International Conference on Learning Representations (ICLR)*, 2023.

[24] J. Lin, J. Tang, H. Tang, S. Yang, X. Dang, and S. Han, "AWQ: Activation-aware weight quantization for LLM compression and acceleration," *arXiv preprint arXiv:2306.00978*, 2023.

[25] Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," arXiv preprint arXiv:2103.13630, 2021.

[26] P. Liang, R. Bommasani, T. Lee *et al.*, "Holistic evaluation of language models," *arXiv preprint arXiv:2211.09110*, 2022.

[27] T. Wolf, L. Debut, V. Sanh *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of EMNLP: System Demonstrations*, 2020.