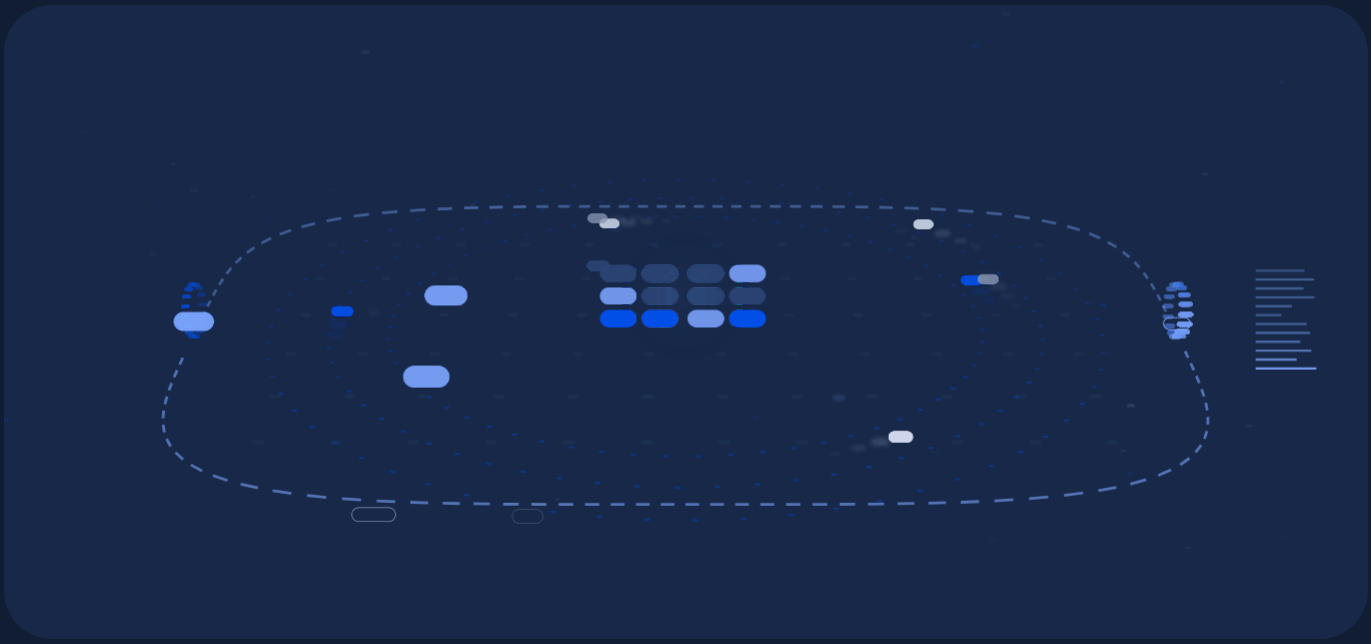
Rillor

METHODS NOTE

# LLM inference benchmarking with TTFT, TPOT and goodput

Comparable LLM inference benchmarking: time to first token and time per output token, each by one formula, goodput within latency limits, and the load and run rules behind each number.



# Contents

01 Summary

02 Background

03 Method

04 Findings

05 Implications

06 Limits of this analysis

07 Sources

# Summary

LLM inference benchmarking gives figures that two serving stacks can be compared on only when time to first token, time per output token and goodput are measured with the same formulas, load and latency limits [1, 9, 14, 20].

- Time to first token (TTFT) is the wait for the first token of a response, and time per output token (TPOT) is the average gap between the tokens after it, in MLPerf's definitions [1]. A TTFT measured at the client also includes server queue time and overhead such as tokenization and transport [9, 14].
- Benchmarking tools use the same metric names with different formulas. GenAI-Perf leaves TTFT out of its inter-token average and LLMPerf includes it [11, 19]. vLLM's documentation advises comparing "the measurement points and formulas rather than the metric names alone" [14].
- Goodput counts only the requests that meet every latency limit. The 2024 DistServe paper defines per-GPU goodput as the "maximum request rate" each GPU serves while a set share of requests, such as 90%, meets the limits. In its example, running prompt processing (prefill) and token generation (decode) on separate GPUs would ideally raise it 2.1 times, from about 1.6 to 3.3 requests per second per GPU [20].
- MLPerf Inference sets latency limits per benchmark at the 99th percentile. For its 70-billion-parameter benchmark they are 2,000 ms TTFT and 200 ms TPOT in the conversational category and 450 ms and 40 ms in the interactive category [1, 4].
- The latency limit changes the throughput figure. One submitter's eight-GPU system in MLPerf® Inference v6.0 sustained 87,444 completed tokens per second under limits of 3,000 ms TTFT and 80 ms TPOT, and 13,155 under 2,000 ms and 15 ms, 15% as many (this note's arithmetic) [5, 6, 7].
- Percentiles carry the tail. In one MLPerf® Inference v6.0 run held to a 3,000 ms TTFT limit, the 99th percentile TTFT was 1,305.9 ms against a median of 143.8 ms, 9.1 times higher (this note's arithmetic) [5]. MLPerf's rules suggest 23,886 queries to measure a 90th percentile at 99% confidence and a 0.50% margin [1].
- Load design moves results. Fixed concurrency and Poisson or gamma arrivals at a set rate load a server differently, a load generator that waits for each reply hides slow periods, and a server that keeps the computed state of earlier prompts (its prefix cache) inflates throughput when a run is repeated [10, 13, 14, 30].
- Research published in 2024 finds that averages and percentiles of TTFT and TPOT miss stalls during generation, and proposes token-level deadlines and a smoothed goodput [23, 28, 33].

# Background

An LLM server answers a request in two phases. Prefill "processes the entire input prompt and produces the first output token", and decode "generates the rest of output tokens, one-at-a-time" [21]. Prefill iterations saturate GPU compute, while decode iterations use little of it [21]. Splitwise describes the token phase as memory-intensive, with its batching limited by memory capacity [22]. That memory holds the KV cache, the keys and values each request keeps for its tokens, which "grows and shrinks dynamically" and limits the batch size [25].

Servers keep many requests in flight at once. Orca replaced request-level batching, under which "newly arrived requests have to wait until the current batch completely finishes", with iteration-level scheduling that runs one model iteration at a time on the batch [24]. NVIDIA's benchmarking guide states the consequence for measurement: "one request's prefill phase can overlap with another request's generation phase" [9]. One request's timings depend on the other requests the server holds.

Overlap creates interference. DistServe's authors write that decoding steps in a batch "are delayed by the prefill steps, significantly elongating their TPOT", and that decoding steps add to TTFT in turn [20]. Chunked prefill with piggybacked decodes, as in SARATHI and DeepSpeed-MII, "trades TTFT for TPOT and cannot eliminate the interference", in DistServe's reading [20]. vLLM enables chunked prefill by default where possible and then "prioritizes decode requests", and its tuning guide says a smaller batched-token budget such as 2,048 gives better ITL and a larger one better TTFT [16].

Benchmarks also disagree for reasons outside the server. NVIDIA's technical blog notes that client-side tools "differ in how they define, measure, and calculate different metrics" [11]. It separates load testing, which studies capacity and scaling under many concurrent requests, from performance benchmarking, which measures throughput, latency and token-level metrics [11]. This note addresses the second, for platform engineers comparing serving engines, GPUs or hosting options.

# Method

**What was examined.** The definitions of time to first token (TTFT), time per output token (TPOT), inter-token latency (ITL), end-to-end latency, throughput and goodput in the MLPerf Inference rules, in five benchmarking tools (vLLM's `vllm bench serve`, SGLang's bench_serving, NVIDIA's AIPerf and GenAI-Perf, and LLMPerf), with their source code where documents leave a formula open, and in the papers that introduced or criticized them. Two MLPerf® Inference v6.0 result logs serve as a worked example.

**Sources.** Thirty-six sources, each opened on 8 November 2026, with [18] opened again and [36] first on 13 November 2026: eight MLCommons documents (the rules, the LoadGen configuration, two articles, two result logs, a system description and a results summary), twelve tool documents and code files (four from NVIDIA, five from vLLM and one each from SGLang, LLMPerf and wrk2), fifteen papers and articles (ten peer-reviewed and five preprints) and one public trace dataset. Several tool documents are "latest" pages, so each Sources entry carries its viewed date, update date or commit.

**How.** Each metric is described from the source that defines it. Figures marked "derived" are this note's arithmetic on a source's own numbers. Model names appear only as the benchmark labels the sources use. No serving engine, tool or GPU is ranked, and every figure comes from a published source.

**Where sources disagree.** MLPerf's rules list a 20 ms TPOT limit for the benchmark they label GPT-OSS-120B in its interactive version, while the LoadGen configuration and a v6.0 log use 15 ms [1, 2, 6]. NVIDIA prefers fixed concurrency to a fixed request rate [10], MLPerf drives load by arrival rate [1], and vLLM offers both and calls an infinite rate with a concurrency cap its most common configuration [13, 14]. Goodput has three working definitions: a maximum rate at an attainment goal such as 90% [20], a rate held at the 99th percentile [1, 8], and a count of requests that met every limit at one offered load [12, 15, 18].

# Findings

## 1. TTFT, TPOT, ITL and end-to-end latency

MLPerf's rules collect two latency metrics for LLM benchmarks: TTFT, "which measures the latency of the first token", and TPOT, "which measures the average interval between all the tokens generated" [1]. NVIDIA's NIM benchmarking guide defines TTFT as "the time from query submission to the first received token, if the response is not empty", and says it "generally includes request queuing time, prefill time, and network latency" [9]. Longer prompts raise it, because attention uses the whole input sequence to build the KV cache [9].

A client measures more than the server does. vLLM's documentation writes client TTFT as roughly server queue time plus server TTFT plus front-end and network overhead, which "includes tokenization, chat-template application, multimodal preprocessing, detokenization, and transport" [14].

ITL is the average gap between consecutive tokens. NVIDIA's guide gives it as another name for TPOT, with ITL = (e2e_latency − TTFT) / (Total_output_tokens − 1) [9]. AIPerf's metrics reference uses the same form and describes ITL as "excluding the initial TTFT overhead" [12]. End-to-end latency is TTFT plus generation time [9].

Throughput comes in two forms. Tokens per second for the whole system "increases until it saturates the available GPU compute resources", and beyond that point it can fall [9]. Tokens per second for each user approaches 1/ITL as outputs lengthen, and as concurrent requests increase, system throughput rises while each user's falls [9]. For reasoning models, AIPerf reports time to first output token, the first non-reasoning token, beside TTFT, which counts "any first token including reasoning tokens" [12].

## 2. How tools define inter-token latency differently

NVIDIA's guide warns that tools "differ on whether TTFT is included in the average", and that AIPerf excludes it [9]. NVIDIA's blog adds that GenAI-Perf leaves TTFT out "as opposed to LLMPerf, which does include the TTFT" [11]. LLMPerf's client code agrees: it stores TTFT as the first entry of its list of token gaps [19]. For a request with a long TTFT and a short output, the two tools report different ITL from identical timings.

vLLM computes TPOT per request as (latency − TTFT) / (output length − 1), and its ITL as the pooled list of every token gap across all requests [15]. A TPOT percentile therefore ranks requests by their average pace, and an ITL percentile ranks single gaps (this note's reading of the code [15]).

Throughput windows differ too. GenAI-Perf divides output tokens by the time from the first request to the last response, while LLMPerf divides by the whole benchmark duration, which includes prompt generation, request preparation and storing responses [11]. NVIDIA's blog says these overheads "in the single concurrency scenario can sometimes account for 33% of the entire benchmark duration" [11]. LLMPerf also sends requests in batches of N and lets them drain, while GenAI-Perf keeps N requests active [11].

Token counts differ as well. vLLM counts output tokens for some back ends by re-tokenizing the returned text, which "may inflate the output token count slightly" [15], and SGLang re-tokenizes too [18]. LLMPerf counts every API's tokens with one fixed tokenizer [19]. AIPerf notes that when a server bundles several tokens into the first streamed chunk, "assuming a single token over-counts the decode tokens and inflates Output Token Throughput Per User" [12].

vLLM's documentation concludes that "Metric terminology is not standardized across benchmarking tools" [14].

## 3. Goodput: throughput within service levels

Throughput alone credits a server for requests it finished late. SGLang's guide says throughput "counts every completed request, including the ones that came back too late to be useful" [18]. DistServe defines per-GPU goodput as "the maximum request rate that can be served adhering to the SLO attainment goal (say, 90%) for each GPU provisioned", and ties it to cost: "higher per-GPU goodput directly translates into lower cost per query" [20]. In its example, a 13-billion-parameter model on one A100 reached about 1.6 requests per second with prefill and decode colocated, and the paper estimates that two prefill GPUs with one decode GPU would ideally reach 3.3 requests per second per GPU, 2.1 times as much [20]. Sarathi-Serve uses the same idea under the name capacity, "the maximum request load (queries-per-second) a system can sustain while meeting certain latency targets" [21].

MLPerf has measured a rate under a latency requirement since its first paper, where the server metric "is the Poisson parameter that indicates the queries-per-second (QPS) achievable while meeting the QoS requirement" [8]. LoadGen finds that rate with a binary search over several runs, and the rules report "Scheduled samples per second" [1].

Tools compute goodput at one offered load. vLLM's `--goodput` takes limits in milliseconds for TTFT, TPOT and end-to-end latency and refers to DistServe for the definition [13]. In its code a request counts as good only if every chosen limit holds, and goodput is good completed requests over the run's duration [15]. SGLang counts a request as good "if it succeeded and met every listed SLO", reports SLO attainment over all requests sent, "so failed requests count as misses", and describes the sweep: "throughput keeps rising or flattens under overload, goodput peaks and then falls" [18]. One tool run gives one point on that curve, and the DistServe or MLPerf figure is the rate a sweep finds (this note's reading of [1, 18, 20]).

Goodput can be gamed. A 2024 study observed that "manually delaying the delivery of some tokens can improve SLOs" and that "actively abandoning requests that do not meet SLOs can improve SLMs", its term for system-level metrics, and proposed a smoothed goodput [28]. Counting failed requests as misses, as SGLang does, keeps abandoned requests in the denominator (this note's reading of [18, 28]).

## 4. Reference service levels from MLPerf Inference

MLPerf's Server scenario sends queries "according to a Poisson distribution" for at least 600 seconds and holds the 99th percentile to a benchmark-specific limit [1, 2]. The Offline scenario has no latency limit [1]. The Interactive versions run as the Server scenario of a separate workload with tighter limits [2, 6].

The rules' datacenter benchmarks table sets TTFT and TPOT limits for six LLM benchmarks, shown with the rules' own labels [1]. The first two rows call their looser category "Conversational", Mixtral-8x7B has one pair of limits with no label, and the others call it "Server".

| Benchmark (rules label) | Conversational or Server limits, TTFT / TPOT | Interactive limits, TTFT / TPOT |
|---|---|---|
| **Llama3.1-8B** | 2,000 ms / 100 ms | 500 ms / 30 ms |
| **Llama2-70b** | 2,000 ms / 200 ms | 450 ms / 40 ms |
| **Llama3.1-405B** | 6,000 ms / 175 ms | 4,500 ms / 80 ms |
| **Mixtral-8x7B** | 2,000 ms / 200 ms | none |
| **DeepSeek-r1** | 2,000 ms / 80 ms | 1,500 ms / 15 ms |
| **GPT-OSS-120B** | 3,000 ms / 80 ms | 2,000 ms / 20 ms |

MLCommons explained the 70B limits in 2025. The original server limits were set "according to the usage models and hardware capabilities of the time, at 2s TTFT and 200 ms TPOT" [4]. The interactive version adopted "a stricter 99th percentile threshold of 25 tokens/second (TPOT of 40ms)" and "a 99th percentile TTFT limit of 450ms" [4]. The MLCommons LLM task force's late-2024 analysis of industry research, user surveys and platform data found a median of 20 to 50 tokens per second critical for a seamless experience. It chose the 99th percentile "To prioritize reliability under peak demand", and noted that "reasoning models and agents are taking advantage of ever lower latencies" [4].

The limits move between files and rounds. The rules list 20 ms TPOT for the GPT-OSS-120B Interactive benchmark, while the LoadGen configuration sets its `tpot_latency` to 15 and a v6.0 result log ran at 15 ms [1, 2, 6]. The v6.1 round, announced on 16 September 2026, drew 30 submitting organizations and allowed speculative decoding in the interactive scenario for two benchmarks [3].

Two logs from one submitter's eight-GPU B200 system in MLPerf® Inference v6.0, both for the benchmark labelled gpt-oss-120b, show the limits at work [5, 6, 7]:

| Figure | Server run | Interactive run |
|---|---|---|
| Limits, TTFT / TPOT | 3,000 ms / 80 ms | 2,000 ms / 15 ms |
| Completed tokens per second (log label) | 87,444.22 | 13,155.34 |
| Completed samples per second | 66.76 | 10.05 |
| TTFT, median / 99th percentile | 143.8 ms / 1,305.9 ms | 62.6 ms / 146.5 ms |
| TPOT, median / 99th percentile | 41.5 ms / 44.8 ms | 6.8 ms / 14.55 ms |
| End-to-end, median / 99th percentile | 52.4 s / 82.2 s | 9.4 s / 23.4 s |
| 99th percentile TTFT as a share of its limit (derived) | 44% | 7% |
| 99th percentile TPOT as a share of its limit (derived) | 56% | 97% |

MLPerf® notes: MLPerf® Inference: Datacenter v6.0, Closed division, Available category, benchmark gpt-oss-120b, Server and Interactive scenarios. Result ID 6.0-0083, which covers both runs. The logs and system description were retrieved on 8 November 2026 and the result ID on 13 November 2026 from the MLCommons results repository [5, 6, 7, 36]. Ratios, percentages and shares of a limit are this note's arithmetic and are not MLPerf® metrics. The MLPerf name and logo are registered and unregistered trademarks of MLCommons Association in the United States and other countries. All rights reserved. Unauthorized use strictly prohibited. See www.mlcommons.org for more information.

Both results are VALID [5, 6]. Under the Interactive limits, TPOT, at 97% of its limit, set the rate (derived) [5, 6]. A TPOT of 15 ms is about 67 tokens per second per request (derived) [6]. MLPerf® results are run by the submitters under common rules, and MLCommons describes its v6.1 results as peer-reviewed [3].

These limits are one benchmark program's reference points for its own workloads [4]. Other sources set other targets. DistServe's evaluation used 0.25 s TTFT and 0.1 s TPOT for a chatbot, 0.125 s and 0.2 s for code completion and 15 s and 0.15 s for summarization [20]. Andes set a 1.3 s TTFT target, which it attributes to guidance for web page loading [33].

## 5. Workload design: prompt and output lengths, arrival process, concurrency

**Lengths.** NVIDIA's parameters guide says longer inputs raise prefill memory and TTFT, and longer outputs raise generation memory and ITL [10]. It gives pairs by use: translation with inputs and outputs of 500 to 2,000 tokens each, generation with about 100 tokens in and 1,000 out, and summarization with about 1,000

in and 100 out [10]. Production traffic varies by service. In a cloud provider's traces from 11 November 2023, coding requests had a median of 1,500 prompt tokens and 13 output tokens, and conversation requests a median of 1,020 prompt tokens with an "almost bimodal" output whose median was 129 [22]. vLLM's random dataset defaults to 1,024 input and 128 output tokens [13]. NVIDIA's guide advises setting `ignore_eos` to true so each request generates to its `max_tokens` limit, while vLLM's `--ignore-eos` defaults to False [10, 13].

**Real text.** Some engine features depend on the words as well as the counts. Automatic prefix caching reuses the KV cache of a shared prefix and "only reduces the time of processing the queries (the prefilling phase)" [17]. Speculative decoding computes several tokens in parallel "without any changes to the outputs", 2 to 3 times faster in its authors' tests [31]. In Spec-Bench, a prompt-lookup method reached about 2.4 times on summarization and 1.1 to 1.3 times on translation and question answering, and speedups moved with sampling temperature [32]. The 2023 trace's notes say prompt text does not affect the metrics its authors benchmark [27]. With prefix caching or speculative decoding on, the text does change the timings (this note's reading of [17, 32]).

**Arrivals.** MLPerf generates requests at a rate, and vLLM can [1, 13]. In `vllm bench serve`, `--request-rate` defaults to inf, which sends all requests at time 0, and a finite rate uses a "Poisson process or gamma distribution to synthesize the request arrival times" [13]. A `--burstiness` of 1 gives a Poisson process, and values between 0 and 1 give burstier traffic [13]. vLLM's documentation gives the coefficient of variation as 1/sqrt(burstiness): about 3.16 at 0.1, 1.0 at 1.0 and about 0.45 at 5.0 [14]. BurstGPT, built from 10.31 million traces of a cloud provider's regional LLM services over 213 days, models burstiness with a Gamma distribution and found that in API services both of its parameters "vary sharply during the day" [26]. vLLM offers a burstgpt dataset and a timed_trace dataset that replays recorded timing [13].

**Concurrency.** NVIDIA's guide prefers fixed concurrency, N clients with one active request each, because with a request rate "outstanding requests can grow without bound when the arrival rate exceeds system throughput" [10]. It suggests sweeping from one request to slightly above the maximum batch size, where throughput saturates while latency keeps rising [10]. With vLLM's `--max-concurrency` and a rate both set, "the actual request rate may be lower than specified" [13], and its documentation calls an infinite rate with a concurrency cap "the most commonly used configuration for production benchmarking" [14]. Patil's 2026 preprint links the controls: the offered rate "sets in-flight concurrency via Little's Law" [35]. The KV cache caps concurrency: in one vLLM startup log, 15,728,640 cached tokens allow 1,920 concurrent requests of 8,192 tokens [14].

## 6. Percentiles, warm-up and run length

MLPerf holds latency at the 99th percentile and sets a minimum run of 600 seconds [1, 2]. Its rules suggest 23,886 queries for the 90th percentile at 99% confidence and a 0.50% margin, and 262,742 for the 99th at a 0.05% margin [1]. An early stopping rule lets shorter runs count, with a penalty: the effective percentile computed for them is slightly higher [1].

The tail differs from the middle. In the Server log, TTFT had a mean of 186.3 ms, a median of 143.8 ms, a 99th percentile of 1,305.9 ms and a maximum of 2,849.2 ms, while TPOT stayed between 41.5 ms at the median and 44.8 ms at the 99th percentile [5]. A mean or median alone would hide the 9.1-fold gap in TTFT (derived) [5]. In the Interactive log, the 99th percentile TPOT of 14.55 ms was about twice the median of 6.8 ms (derived) [6].

Tails compound when one task makes many calls. In Dean and Barroso's example, a server that typically responds in 10 ms and takes one second at the 99th percentile makes 63% of user requests take more than a second when each request needs 100 such servers in parallel [29]. By the same arithmetic, a task of 20 independent model calls, each with a 1% chance of being slow, meets at least one slow call 18% of the time (derived, 1 − 0.99^20, assuming independence) [29].

How the load generator times requests decides what the tail shows. A generator that sends a request only after the previous reply "exhibits a strong Coordinated Omission effect", and wrk2 instead measures from "the time the transmission *should* have occurred" [30]. MLPerf defines latency from "when the LoadGen was scheduled to pass a query to the SUT" [1]. vLLM records client queue time, the wait for a concurrency slot, separately, and at a finite rate offers `e2el_including_client_queue` for schedule-relative latency [14, 15]. In the v6.0 Server log, completed samples per second were 96% of scheduled (derived) [5].

Warm-up defaults differ. vLLM sends 0 warm-up requests [13], SGLang 1 [18], GenAI-Perf leaves out "warming up" and "cooling down" requests with a sliding window [11], and AIPerf excludes a configured warm-up phase [9]. Repeating `vllm bench serve` can reuse prompts left in the prefix cache and inflate throughput, since the random dataset repeats for a fixed `--seed` [14]. With `--flush-cache`, SGLang's benchmark resets a vLLM server's prefix cache after warm-up, and if that server was not started with `VLLM_SERVER_DEV_MODE=1` the benchmark "fails loudly rather than accidentally measuring warm-cache performance" [18].

## 7. Critiques of TTFT and TPOT

Etalon's authors argue that the standard metrics hide what users see [23]. TTFT "is oblivious of prompt length", so a static TTFT limit is "not practical" [23]. In their tests, one system kept almost 60% of requests waiting more than 25 seconds against a maximum of 15 seconds in another, while normalized latency differed by a few hundred milliseconds [23]. TPOT hides "jitters in token generation", since a 10-second stall barely moves a request's average [23]. They propose a fluidity-index, the share of token-level deadlines met, with an example target of 99% of requests meeting deadlines at least 90% of the time [23].

Andes finds that "simple statistics derived from a subset of token delivery timestamps", averages and 90th or 99th percentiles of TPOT or TTFT among them, "fail to fully capture user experience" [33]. Under load surges it observed TTFT "significantly inflated due to head-of-line blocking" [33]. Sarathi-Serve calls such pauses "generation stalls", and its stall-free schedules with chunked prefills target them [21].

How fast tokens need to arrive depends on who reads them. DistServe notes that for chatbots TPOT "only remains important until it is faster than human reading speed (i.e., 250 words/min)" [20]. Etalon equates that reading speed with about 6 tokens per second [23], and Andes uses reading and listening speeds of 4.8

and 3.3 tokens per second [33]. MLPerf's interactive limit of 40 ms, 25 tokens per second, sits above those figures [4].

The tools offer partial answers. SGLang reports ITL percentiles and the maximum gap beside TPOT [18], and vLLM can send probe requests, "Useful for measuring how the main workload stalls unrelated requests" [13].

## 8. A protocol for comparing two serving stacks

1. **Set the limits first.** Choose TTFT and TPOT limits from the product's use, since chat, code completion and summarization call for different ones [20, 22], or adopt MLPerf's limits for a named benchmark and cite the file and date [1, 2].
2. **Fix the tool and its formulas.** Run one tool version against both stacks, and record how it computes TTFT, TPOT, ITL and throughput [9, 11, 14].
3. **Record every version and setting.** MLPerf's Network division requires its "Available" submissions to "specify software version of all components, hardware configurations, software stacks, dockers, and settings of all components and stacks" [1]. One batching setting trades ITL against TTFT [16]. Vidur's authors estimated in 2024 that exploring configurations for a 70B model by deployment would take 42,000 GPU hours, about $218,000 [34], so the configurations to test are chosen before the runs.
4. **Build the workload from real traffic.** Use the deployment's input and output length distributions, or replay a public trace with its timing [13, 22, 27]. Use real text when prefix caching or speculative decoding is on [17, 32].
5. **Drive load and time it from the schedule.** Use arrival rates with latency measured from the scheduled send time, or a concurrency cap with client queue time recorded [1, 10, 14, 30]. Test Poisson and burstier arrivals [13, 14, 26].
6. **Control warm-up, caches and run length.** Leave warm-up requests out, flush or vary the prefix cache between runs, and run long enough for the percentile reported [1, 11, 14, 18].
7. **Sweep the rate and report goodput.** Raise the request rate from light load past saturation, and record goodput and SLO attainment at the chosen limits, with failed requests counted as misses [18, 28].
8. **Report distributions.** Publish the median, 90th and 99th percentiles and maximum for TTFT, TPOT, ITL and end-to-end latency, and compare results only "when definitions align" [9, 23].

# Implications

**For platform engineers.** A benchmark result is a rate together with the limits it was held to. The v6.0 logs show one system at 87,444 and 13,155 completed tokens per second under two sets of limits [5, 6], so a team sets its TTFT and TPOT limits from the product before it measures [4, 20]. The KV cache bounds how many requests each GPU holds [14, 25], which ties benchmark design to memory sizing, the subject of How much GPU memory LLM inference needs (https://rillor.com/insights/gpu-memory-for-llm-inference).

**For buyers comparing published figures.** A tokens-per-second figure without its latency limits, percentile, tool, prompt lengths and arrival pattern cannot be set beside another [1, 9, 14]. Useful questions: which percentile, which formulas, which load control, whether failed requests counted and whether runs started warm [1, 11, 14, 18].

**For cost estimates.** Goodput per GPU is the throughput that belongs in a cost model, because in DistServe's words it "directly translates into lower cost per query" [20]. The break-even arithmetic that uses throughput within a latency limit is in Self-hosted LLM vs API cost: a break-even method (https://rillor.com/insights/self-hosted-llm-vs-api-cost).

**For agent workloads.** A task that makes many model calls meets the tail more often [29], and MLCommons names agents among the uses behind its tighter limits [4].

**For private deployments.** In a private superintelligence (SI) cloud (https://rillor.com/compute/private-si#runs) that Rillor deploys and operates, models and agents are tested on the client's tasks before release, and health, capacity and model latency are watched on the platform.

# Limits of this analysis

- Documentation changes. vLLM's pages are its "latest developer preview docs" [13], and other tool pages were updated within weeks of viewing [9, 12, 18]. Defaults quoted here held on 8 November 2026.
- MLPerf's limits change by round, and the rules and the LoadGen configuration disagree on one interactive TPOT limit [1, 2]. The worked example uses one system's two v6.0 logs for one benchmark and uses no v6.1 results [3, 5, 6].
- The serving research ran on 2022 to 2024 systems. Sarathi-Serve's gains of 2.6 to 3.7 times were measured against the vLLM of 2024, before chunked prefill became its default [16, 21].
- The production traces come from one cloud provider [22, 26, 27]. The speculative decoding speedups come from a 7B model on one consumer GPU and from the original paper's tests [31, 32].
- The tail arithmetic assumes independent calls [29]. Load surges make slow periods arrive together [33].
- The TTFT and TPOT targets cited are each source's own [1, 4, 20, 23, 33]. None of the sources sets one threshold for every use.
- This note covers serving speed. Output quality and output variance under load are separate questions.

# Sources


1. MLPerf Inference Rules (inference_rules.adoc). MLCommons, GitHub mlcommons/inference_policies, last changed 20 August 2026 (commit ff7edba545), viewed 8 November 2026. https://raw.githubusercontent.com/mlcommons/inference_policies/master/inference_rules.adoc

2. MLPerf LoadGen configuration (loadgen/mlperf.conf). MLCommons, GitHub mlcommons/inference, last changed 2 July 2026 (commit b0662794362b), viewed 8 November 2026. https://raw.githubusercontent.com/mlcommons/inference/master/loadgen/mlperf.conf

3. MLCommons Sets Participation Record with New MLPerf Inference v6.1 Benchmark Results. MLCommons, 16 September 2026, viewed 8 November 2026. https://mlcommons.org/2026/09/mlperf-inference-v6-1-results/

4. MLPerf Inference v5.0 Advances Language Model Capabilities for GenAI. Thomas Atta-Fosu, Viraat Chandra, Miro Hodak and others, MLCommons, 2 April 2025, viewed 8 November 2026. https://mlcommons.org/2025/04/llm-inference-v5/

5. MLPerf Inference v6.0, closed division: 8x B200 system, gpt-oss-120b, Server, performance log summary. MLCommons, GitHub mlcommons/inference_results_v6.0, 2026, viewed 8 November 2026. https://raw.githubusercontent.com/mlcommons/inference_results_v6.0/main/closed/Nebius/results/nebius_b200_n1/gpt-oss-120b/Server/performance/run_1/mlperf_log_summary.txt

6. MLPerf Inference v6.0, closed division: 8x B200 system, gpt-oss-120b, Interactive, performance log summary. MLCommons, GitHub mlcommons/inference_results_v6.0, 2026, viewed 8 November 2026. https://raw.githubusercontent.com/mlcommons/inference_results_v6.0/main/closed/Nebius/results/nebius_b200_n1/gpt-oss-120b/Interactive/performance/run_1/mlperf_log_summary.txt

7. MLPerf Inference v6.0, closed division: 8x B200 system, system description. MLCommons, GitHub mlcommons/inference_results_v6.0, 2026, viewed 8 November 2026. https://raw.githubusercontent.com/mlcommons/inference_results_v6.0/main/closed/Nebius/systems/nebius_b200_n1.json

8. MLPerf Inference Benchmark. Vijay Janapa Reddi and others, ISCA 2020, arXiv:1911.02549, version 2, 9 May 2020, viewed 8 November 2026. https://arxiv.org/abs/1911.02549

9. LLM Inference Benchmarking: Metrics. NVIDIA NIM Benchmarking documentation, LLM 2.0.0, NVIDIA, updated 5 October 2026, viewed 8 November 2026. https://docs.nvidia.com/nim/benchmarking/llm/latest/metrics.html

10. LLM Inference Benchmarking: Parameters. NVIDIA NIM Benchmarking documentation, NVIDIA, updated 10 September 2026, viewed 8 November 2026. https://docs.nvidia.com/nim/benchmarking/llm/latest/parameters.html

11. LLM Inference Benchmarking: Fundamental Concepts. Vinh Nguyen, Wenwen Gao, Emily Apsey and others, NVIDIA Technical Blog, 2 April 2025, viewed 8 November 2026. https://developer.nvidia.com/blog/llm-benchmarking-fundamental-concepts/

12. AIPerf Metrics Reference (docs/metrics-reference.md). NVIDIA, GitHub ai-dynamo/aiperf, last changed 5 November 2026 (commit cacb6f8fb181), viewed 8 November 2026. https://raw.githubusercontent.com/ai-dynamo/aiperf/main/docs/metrics-reference.md

13. vllm bench serve. vLLM documentation, latest developer preview, vLLM project, viewed 8 November 2026. https://docs.vllm.ai/en/latest/cli/bench/serve.html

14. Benchmark CLI. vLLM documentation, latest, vLLM project, viewed 8 November 2026. https://docs.vllm.ai/en/latest/benchmarking/cli.html

15. vllm/benchmarks/serve.py. vLLM project, GitHub vllm-project/vllm, last changed 7 November 2026 (commit 690364833e80), viewed 8 November 2026. https://raw.githubusercontent.com/vllm-project/vllm/main/vllm/benchmarks/serve.py

16. Optimization and Tuning. vLLM documentation, latest, vLLM project, viewed 8 November 2026. https://docs.vllm.ai/en/latest/configuration/optimization.html

17. Automatic Prefix Caching. vLLM documentation, latest, vLLM project, viewed 8 November 2026. https://docs.vllm.ai/en/latest/features/automatic_prefix_caching.html

18. Bench Serving Guide. SGLang documentation, SGLang project, updated 12 November 2026, viewed 13 November 2026. https://docs.sglang.io/docs/developer_guide/bench_serving

19. LLMPerf, README and chat completions client (src/llmperf/ray_clients/openai_chat_completions_client.py). Anyscale, GitHub ray-project/llmperf, archived, last pushed 9 December 2024, viewed 8 November 2026. https://raw.githubusercontent.com/ray-project/llmperf/main/README.md and https://raw.githubusercontent.com/ray-project/llmperf/main/src/llmperf/ray_clients/openai_chat_completions_client.py

20. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. Yinmin Zhong, Shengyu Liu, Junda Chen and others, OSDI 2024, arXiv:2401.09670, version 3, 6 June 2024, viewed 8 November 2026. https://arxiv.org/abs/2401.09670

21. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. Amey Agrawal, Nitin Kedia, Ashish Panwar and others, OSDI 2024, arXiv:2403.02310, version 3, 17 June 2024, viewed 8 November 2026. https://arxiv.org/abs/2403.02310

22. Splitwise: Efficient generative LLM inference using phase splitting. Pratyush Patel, Esha Choukse, Chaojie Zhang and others, ISCA 2024, arXiv:2311.18677, version 2, 20 May 2024, viewed 8 November 2026. https://arxiv.org/abs/2311.18677

23. Etalon: Holistic Performance Evaluation Framework for LLM Inference Systems. Amey Agrawal, Anmol Agarwal, Nitin Kedia and others, arXiv:2407.07000, version 2, 30 August 2024, viewed 8 November 2026. https://arxiv.org/abs/2407.07000

24. Orca: A Distributed Serving System for Transformer-Based Generative Models. Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim and Byung-Gon Chun, OSDI 2022, USENIX, July 2022, viewed 8 November 2026. https://www.usenix.org/system/files/osdi22-yu.pdf

25. Efficient Memory Management for Large Language Model Serving with PagedAttention. Woosuk Kwon, Zhuohan Li, Siyuan Zhuang and others, SOSP 2023, arXiv:2309.06180, 12 September 2023, viewed 8 November 2026. https://arxiv.org/abs/2309.06180

26. BurstGPT: A Real-world Workload Dataset to Optimize LLM Serving Systems. Yuxin Wang, Yuhan Chen, Zeyu Li and others,

arXiv:2401.17644, version 5, 26 May 2025, viewed 8 November 2026. https://arxiv.org/abs/2401.17644

27. Azure LLM inference trace 2023 (AzureLLMInferenceDataset2023.md). Microsoft Azure, GitHub Azure/AzurePublicDataset, collected 11 November 2023, viewed 8 November 2026. https://raw.githubusercontent.com/Azure/AzurePublicDataset/master/AzureLLMInferenceDataset2023.md

28. Revisiting Service Level Objectives and System Level Metrics in Large Language Model Serving. Zhibin Wang, Shipeng Li, Yuhang Zhou and others, arXiv:2410.14257, version 2, 29 October 2025, viewed 8 November 2026. https://arxiv.org/abs/2410.14257

29. The Tail at Scale. Jeffrey Dean and Luiz André Barroso, Communications of the ACM 56(2), February 2013, pp. 74-80, viewed 8 November 2026. https://www.barroso.org/publications/TheTailAtScale.pdf

30. wrk2, README. Gil Tene, GitHub giltene/wrk2, viewed 8 November 2026. https://raw.githubusercontent.com/giltene/wrk2/master/README.md

31. Fast Inference from Transformers via Speculative Decoding. Yaniv Leviathan, Matan Kalman and Yossi Matias, ICML 2023, arXiv:2211.17192, viewed 8 November 2026. https://arxiv.org/abs/2211.17192

32. Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding. Heming Xia, Zhe Yang, Qingxiu Dong and others, Findings of ACL 2024, arXiv:2401.07851, version 3, 4 June 2024, viewed 8 November 2026. https://arxiv.org/abs/2401.07851

33. Andes: Defining and Enhancing Quality-of-Experience in LLM-Based Text Streaming Services. Jiachen Liu, Jae-Won Chung, Zhiyu Wu and others, arXiv:2404.16283, version 2, 13 December 2024, viewed 8 November 2026. https://arxiv.org/abs/2404.16283

34. Vidur: A Large-Scale Simulation Framework For LLM Inference. Amey Agrawal, Nitin Kedia, Jayashree Mohan and others, MLSys 2024, arXiv:2405.05465, version 2, 21 May 2024, viewed 8 November 2026. https://arxiv.org/abs/2405.05465

35. Beyond Per-Token Pricing: A Concurrency-Aware Methodology for LLM Infrastructure Cost Estimation. Chitral Patil, arXiv:2606.11690, 10 June 2026, viewed 8 November 2026. https://arxiv.org/abs/2606.11690

36. MLPerf Inference v6.0 results (summary_results.json). MLCommons, GitHub mlcommons/inference_results_v6.0, 2026, viewed 13 November 2026. https://raw.githubusercontent.com/mlcommons/inference_results_v6.0/main/summary_results.json

## About this report

Published by Rillor on 14 November 2026. Superintelligence (SI) tools assist research and drafting, and every figure cites its source. Rillor builds and secures SI systems, supplies the data they learn from and delivers the infrastructure they run on. We apply SI to financial and compute markets.

Cite as: Rillor. LLM inference benchmarking with TTFT, TPOT and goodput. 14 November 2026. https://rillor.com/insights/llm-inference-benchmarking-ttft-tpot

## Notices

Rillor is not a registered investment adviser or commodity trading advisor and does not provide investment or trading advice. Research, forecasts, data and software described on this site are for research and engineering use. Nothing here is an offer or recommendation to buy or sell any security, commodity interest or digital asset. Past or simulated results do not indicate future results.

NVIDIA, AMD, Ceph, Kubernetes and related product names are trademarks of their owners and are used only to identify products. No affiliation or endorsement is implied.

Superintelligence (SI) outputs can be wrong. Important decisions should include human review.

## Work with Rillor

Send us the workloads you want to run, the data they use, who uses them and where your hardware sits today. Contact us at rillor.com/contact?type=private-si or contact@rillor.com. More on private SI cloud: rillor.com/compute/private-si