Network Working Group
Internet-Draft
Intended status: Informational
Expires: 24 July 2026

M. Gaikwad
Independent Researcher

January 2026

Benchmarking Terminology for Large Language Model Serving
draft-gaikwad-llm-benchmarking-terminology-00

Abstract

This document defines terminology for benchmarking the performance of Large Language Model (LLM) inference serving systems. It establishes a shared vocabulary for latency, throughput, resource utilization, and quality metrics applicable to inference engines, application gateways, and compound agentic systems. This document defines terminology only and does not prescribe benchmark methodologies or acceptance thresholds.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 5 July 2026.

Copyright Notice

Table of Contents

1. Introduction

Large Language Model inference serving has emerged as a distinct category of network service with performance characteristics unlike traditional request-response systems. The autoregressive generation process produces output tokens sequentially, creating a streaming response pattern where latency has multiple meaningful definitions. The prefill and decode phases exhibit different computational profiles. Memory consumption grows with sequence length due to key-value cache requirements.

Large Language Model serving systems are increasingly deployed as Internet-facing services, often exposed via standardized APIs and shared infrastructure. Their performance characteristics influence availability, fairness, and side-channel risk in multi-tenant environments. Establishing consistent benchmarking terminology enables clearer communication among implementers, operators, and researchers.

Despite widespread deployment of LLM serving systems, no standard terminology exists for describing their performance. Different implementations, benchmarks, and academic publications use inconsistent definitions for terms such as "throughput," "latency," and "tokens per second." This inconsistency hinders meaningful comparison across systems and creates confusion for practitioners.

This document addresses the terminology gap by providing precise definitions for LLM serving performance metrics. The structure and approach follow [RFC2647], which established benchmarking terminology for firewall performance. Each term includes a definition, discussion of context and implementation considerations, unit of measurement, open issues, and cross-references to related terms.

This document defines terminology only. It does not specify benchmark methodologies, workload profiles, or acceptance criteria. Companion documents may address those topics.

The metrics in this document apply to transformer-based autoregressive language models. Other model architectures such as diffusion models, encoder-only models, or non-autoregressive decoders require different terminology not covered here.

2. Scope and System Under Test Boundary

A prerequisite for benchmarking is defining the System Under Test (SUT) boundary. The same metric measured at different boundaries yields different values. This document identifies three SUT boundary categories:

Model Engine: The inference runtime executing model forward passes. This boundary excludes network transport, authentication, request routing, and safety filtering. Metrics at this boundary reflect raw inference capability.

Application Gateway: The model engine plus request handling, authentication, rate limiting, input validation, output filtering, and safety mechanisms. Metrics at this boundary reflect the performance users observe when calling an API endpoint.

Compound System: An application gateway plus orchestration logic, retrieval components, tool execution, and multi-step reasoning. Metrics at this boundary reflect end-to-end task completion performance for agentic or retrieval-augmented workloads.

Testers MUST declare the SUT boundary when reporting metrics. Metrics from different SUT boundaries MUST NOT be directly compared without adjustment.

The measurement point within the SUT also affects results. For latency metrics, testers MUST specify whether measurement occurs at the client, at the network edge, or within the serving infrastructure.

3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Terminology

4.1. Request and Response Timing Metrics

4.1.1. End-to-End Latency

Definition: The elapsed time between request initiation by a client and receipt of the complete response by that client.

Discussion: End-to-end latency encompasses all processing stages: network transmission, queuing, prefill computation, decode iterations, output filtering, and return transmission. For streaming responses, end-to-end latency is measured until the final token is received.

End-to-end latency depends on output length. Longer responses require more decode iterations and produce higher latency. When comparing systems, testers SHOULD control for output length or report latency normalized by output token count.

Client-side measurement includes network round-trip time. Server-side measurement excludes external network latency but may still include internal network hops within a distributed serving system.

Unit of measurement: milliseconds (ms) or seconds (s)

Issues:

* Measurement point (client-side vs server-side)

* Treatment of failed or truncated responses

* Clock synchronization for distributed measurement

See also: Time to First Token (Section 4.1.2), Time per Output Token (Section 4.1.5), Decode Latency (Section 4.2.2)

4.1.2. Time to First Token

Definition: The elapsed time between request initiation and receipt of the first output token.

Discussion: Time to First Token (TTFT) measures how long a user waits before any response content appears. For interactive applications, TTFT determines perceived responsiveness independent of total response length.

TTFT includes network transmission, authentication, admission control, queue wait time, and prefill computation. Under low load with short prompts, prefill dominates TTFT. Under high load, queue wait time may dominate. With long prompts, prefill computation scales with input token count.

For non-streaming responses, TTFT equals end-to-end latency because all tokens arrive together. Testers SHOULD specify whether the response mode is streaming or non-streaming.

Some systems emit an empty or whitespace-only first token before substantive content. Testers MUST specify whether TTFT measures time to any token or time to first non-empty token.

Unit of measurement: milliseconds (ms)

Issues:

* Streaming vs non-streaming response modes
* Definition of "first token" when initial tokens lack content
* Client-side vs server-side measurement point

See also: Prefill Latency (Section 4.2.1), Queue Wait Time (Section 4.5.3), End-to-End Latency (Section 4.1.1)

4.1.3. Inter-Token Latency

Definition: The elapsed time between consecutive output token emissions, measured at the server.

Discussion: Inter-Token Latency (ITL) measures the generation interval between adjacent tokens during the decode phase. ITL reflects decode efficiency and is affected by batch size, model architecture, and memory bandwidth.

ITL varies across tokens within a single request due to batching dynamics. When other requests join or leave the batch, the per-request compute allocation changes. Testers SHOULD report ITL distribution statistics rather than a single value.

ITL is measured server-side and excludes network transmission delay. For client-observed intervals, see Time Between Tokens (Section 4.1.4).

When aggregating ITL across requests, token-weighted averaging counts each token equally. Request-weighted averaging counts each request equally regardless of length. These methods yield different results. Testers MUST specify the aggregation method.

Unit of measurement: milliseconds (ms)

Issues:

* Token-weighted vs request-weighted aggregation
* Variation within a single request due to batching
* Exclusion of prefill-to-first-token interval

See also: Time Between Tokens (Section 4.1.4), Time per Output Token (Section 4.1.5), Batch Utilization (Section 4.5.5)

4.1.4. Time Between Tokens

Definition: The elapsed time between receipt of consecutive output tokens at the client.

Discussion: Time Between Tokens (TBT) measures the client-observed interval between adjacent tokens. TBT equals ITL plus network transmission variability and buffering effects.

Network jitter, TCP buffering, and intermediary proxies cause TBT to differ from ITL. Multiple tokens may arrive in a single network packet, producing near-zero TBT followed by a longer gap.

TBT directly affects user-perceived streaming smoothness. High TBT variance creates a "stuttering" appearance even when average TBT is acceptable.

Unit of measurement: milliseconds (ms)

Issues:

* Network buffering causing token bunching

* Proxy and CDN effects on delivery timing

* Measurement requires client-side instrumentation

See also: Inter-Token Latency (Section 4.1.3), Token Delivery Jitter (Section 4.4.3)

4.1.5. Time per Output Token

Definition: The average time to generate each output token after the first token, computed as:

TPOT = (End-to-End Latency - TTFT) / (Output Token Count - 1)

Discussion: Time per Output Token (TPOT) summarizes decode-phase performance in a single value. Unlike ITL, which measures each interval, TPOT averages across all decode steps for a request.

TPOT excludes the first token because TTFT captures that interval separately. For single-token outputs, TPOT is undefined.

TPOT is request-weighted by construction: each request contributes one TPOT value regardless of output length. When aggregating across requests, report the distribution rather than only the mean.

TPOT relates to user-perceived generation speed. A TPOT of 50ms corresponds to 20 tokens per second, approximately 900 words per minute for English text.

Unit of measurement: milliseconds per token (ms/token)

Issues:

* Undefined for single-token outputs

* Request-weighted aggregation differs from token-weighted ITL

* Denominator uses (Output Token Count - 1)

See also: Inter-Token Latency (Section 4.1.3), End-to-End Latency (Section 4.1.1), Time to First Token (Section 4.1.2)

4.1.6. Normalized Latency

Definition: End-to-end latency divided by output token count, yielding a length-independent latency measure.

Discussion: Normalized latency enables comparison across requests

with different output lengths. It amortizes fixed overhead (TTFT) across all tokens.

For short outputs, TTFT dominates normalized latency. For long outputs, TPOT dominates. Testers SHOULD report output length distribution alongside normalized latency to enable interpretation.

Normalized latency obscures user-facing behavior because users experience TTFT and TPOT separately. Testers SHOULD NOT report normalized latency as the sole latency metric.

Unit of measurement: milliseconds per token (ms/token)

Issues:

* Obscures distinction between TTFT and decode latency
* Sensitive to output length distribution
* Not directly interpretable as user experience

See also: End-to-End Latency (Section 4.1.1), Time per Output Token (Section 4.1.5)

4.2. Phase-Specific Latency Metrics

4.2.1. Prefill Latency

Definition: The elapsed time to process input tokens and compute the initial key-value cache prior to generating the first output token.

Discussion: Prefill performs a forward pass over all input tokens to populate the key-value cache. This computation is parallelizable across the input sequence and is compute-bound on current hardware.

Prefill latency scales approximately linearly with input token count for uncached inputs. With prefix caching enabled, prefill processes only the uncached suffix, reducing latency for requests sharing common prefixes.

Prefill latency is a component of TTFT. Other TTFT components include queue wait time, network latency, and authentication overhead.

Chunked prefill implementations split long prefills into smaller segments interleaved with decode steps. This reduces head-of-line blocking but increases total prefill time. Testers MUST specify whether chunked prefill is enabled.

Unit of measurement: milliseconds (ms)

Issues:
* Chunked vs monolithic prefill execution
* Prefix caching effects on effective prefill length
* Distinction from TTFT in reporting

See also: Time to First Token (Section 4.1.2), Prefix Cache Hit Rate (Section 4.7.1), Head-of-Line Blocking (Section 4.5.1)

4.2.2. Decode Latency

Definition: The cumulative elapsed time spent generating output tokens after the first token.

Discussion: Decode latency equals (Output Token Count - 1) multiplied by average ITL. It measures total time in the decode phase.

Decode is memory-bandwidth-bound on current hardware because each step reads the full model weights and growing key-value cache while producing a single token.

Decode latency grows linearly with output token count under constant batching conditions. Variable batch membership during generation causes decode latency to deviate from simple linear scaling.

Unit of measurement: milliseconds (ms) or seconds (s)

Issues:
* Variation due to dynamic batching
* Growth of key-value cache during decode

See also: Inter-Token Latency (Section 4.1.3), End-to-End Latency (Section 4.1.1), Prefill Latency (Section 4.2.1)

4.3. Throughput and Capacity Metrics

4.3.1. Output Token Throughput

Definition: The number of output tokens generated per second by the system across all concurrent requests.

Discussion: Output token throughput measures system-wide generation capacity. It increases with offered load until the system saturates, then plateaus or declines.

Throughput measurement requires specifying the token counting method. Subword tokenization produces different counts than word-level tokenization for the same text. Testers MUST identify the tokenizer used.

Some implementations pad sequences to uniform length within a batch. Testers MUST specify whether throughput counts include or exclude padding tokens.

Throughput measurement requires a defined time window. Short windows capture instantaneous throughput. Longer windows smooth over load variations. Testers MUST specify the measurement window duration.

Unit of measurement: tokens per second (tok/s)

Issues:

* Tokenizer-dependent token counts
* Inclusion or exclusion of padding tokens
* Measurement window duration

See also: Input Token Throughput (Section 4.3.2), Request Throughput (Section 4.3.3), Non-Padding Token Throughput (Section 4.3.4)

4.3.2. Input Token Throughput

Definition: The number of input tokens processed per second by the system across all concurrent requests.

Discussion: Input token throughput measures prefill capacity. Systems with efficient batched prefill achieve higher input throughput than those processing prompts sequentially.

Input throughput differs from output throughput because prefill and decode have different computational characteristics. A system optimized for long-context prefill may show high input throughput but lower output throughput, and vice versa.

Prefix caching affects input throughput measurement. With caching, input tokens divide into cache hits (not processed) and cache misses (processed). Testers MUST specify whether input throughput counts all input tokens or only cache misses.

Unit of measurement: tokens per second (tok/s)

Issues:
* Treatment of cached vs processed input tokens
* Different from output throughput due to phase characteristics

See also: Output Token Throughput (Section 4.3.1), Prefill Latency (Section 4.2.1), Prefix Cache Hit Rate (Section 4.7.1)

4.3.3. Request Throughput

Definition: The number of requests completed per second.

Discussion: Request throughput counts completed requests regardless of token counts. A system completing many short requests achieves higher request throughput than one completing fewer long requests, even at equal token throughput.

Request throughput is relevant for capacity planning when requests have predictable lengths or when per-request overhead dominates.

Failed requests require specified handling. Testers MUST specify whether request throughput includes only successful completions or also counts failures.

Unit of measurement: requests per second (req/s)

Issues:
* Treatment of failed or truncated requests
* Sensitivity to request length distribution

See also: Output Token Throughput (Section 4.3.1), End-to-End Latency (Section 4.1.1)

4.3.4. Non-Padding Token Throughput

Definition: Output token throughput excluding padding or alignment tokens.

Discussion: Batched inference pads sequences to uniform length. Padding tokens consume compute but carry no information. Non-padding throughput measures useful work.

Systems using variable-length batching or continuous batching may avoid padding entirely. For these systems, non-padding throughput equals total output throughput.

Unit of measurement: tokens per second (tok/s)

Issues:
* Applicable only to padded batching schemes

See also: Output Token Throughput (Section 4.3.1), Batch Utilization (Section 4.5.5)

4.3.5. Offered Load

Definition: The request arrival rate or concurrency level imposed on the system by the workload generator.

Discussion: Offered load characterizes workload intensity. Two forms exist:

Open-loop load specifies a request arrival rate (requests per second) independent of system response. New requests arrive according to a specified distribution regardless of outstanding request count.

Closed-loop load specifies a fixed concurrency level. Each completed request triggers a new request, maintaining constant outstanding requests.

Open-loop load reveals system behavior under overload. Closed-loop load cannot exceed system capacity by construction. Testers MUST specify the load model and its parameters.

Unit of measurement: requests per second (req/s) for open-loop; concurrent requests for closed-loop

Issues:
* Open-loop vs closed-loop model selection

* Arrival distribution for open-loop (Poisson, uniform, bursty)

* Think time between requests for closed-loop

See also: Sustainable Load (Section 4.3.6), Queue Depth (Section 4.5.2)

4.3.6. Sustainable Load

Definition: The maximum offered load at which the system continues to meet declared service objectives.

Discussion: Sustainable load identifies the operating region boundary. Below sustainable load, the system meets latency and quality targets. Above sustainable load, latency increases unboundedly or requests fail.

Sustainable load depends on the service objectives. Stricter latency targets yield lower sustainable load. Testers MUST declare the service objectives when reporting sustainable load.

Sustainable load also depends on workload characteristics. Longer prompts or outputs reduce sustainable load. Testers MUST characterize the workload profile.

Unit of measurement: requests per second (req/s) or concurrent requests

Issues:

* Requires declared service objectives

* Sensitive to workload characteristics

* May differ for different SLO percentiles

See also: Offered Load (Section 4.3.5), Service Level Objective (Section 4.12.1), SLO Attainment Rate (Section 4.12.2)

4.4. Latency Distribution Metrics

4.4.1. Latency Percentiles

Definition: Values below which a specified percentage of observations fall, reported as P50, P90, P95, P99, and P99.9 for latency metrics.

Discussion: Mean latency obscures distribution shape. A system with

low mean but high variance provides inconsistent user experience. Percentiles characterize the distribution.

P50 (median) indicates typical experience. P99 indicates worst-case experience for most users. P99.9 indicates extreme tail behavior relevant for high-volume services.

Percentile computation requires sufficient sample size. For P99 accuracy, at least 1000 samples are needed. For P99.9, at least 10000 samples. Testers MUST report sample size alongside percentiles.

Percentiles apply to TTFT, TPOT, ITL, and end-to-end latency. Testers SHOULD report percentiles for multiple metrics rather than a single summary.

Unit of measurement: Same as the underlying latency metric (ms)

Issues:

* Sample size requirements for tail percentiles
* Appropriate percentile selection for use case
* Confidence intervals for reported percentiles

See also: End-to-End Latency (Section 4.1.1), Time to First Token (Section 4.1.2), Time per Output Token (Section 4.1.5)

4.4.2. Latency Distribution

Definition: The complete statistical distribution of latency observations, represented as a histogram or cumulative distribution function.

Discussion: Full distributions reveal structure that percentiles miss. Multimodal distributions indicate distinct operating regimes. Heavy tails indicate outlier sensitivity.

Histogram bin width affects resolution. Narrow bins reveal detail but require more samples. Testers SHOULD use logarithmic binning for latency distributions spanning multiple orders of magnitude.

Unit of measurement: Histogram counts or cumulative probability

Issues:

* Bin width selection
* Sample size for distribution estimation

See also: Latency Percentiles (Section 4.4.1)

4.4.3. Token Delivery Jitter

Definition: The variance or standard deviation of inter-token intervals within a single request.

Discussion: Jitter measures streaming smoothness. Low jitter indicates consistent token pacing. High jitter indicates irregular delivery that users perceive as stuttering.

Jitter arises from batching dynamics, memory bandwidth contention, and garbage collection pauses. Systems with continuous batching show higher jitter than static batching due to variable batch membership.

Jitter is computed per-request, then aggregated. Report the distribution of per-request jitter values.

Unit of measurement: milliseconds (ms) as standard deviation

Issues:

* Aggregation method across requests
* Separating server-side jitter from network jitter

See also: Inter-Token Latency (Section 4.1.3), Time Between Tokens (Section 4.1.4)

4.4.4. Maximum Pause Duration

Definition: The longest inter-token interval observed within a single request.

Discussion: Maximum pause captures the worst interruption in streaming output. A single long pause degrades user experience even when average ITL is acceptable.

Long pauses arise from garbage collection, KV cache operations, batch recomputation after preemption, and request scheduling delays.

Unit of measurement: milliseconds (ms)

Issues:

* Distinguishing generation pauses from network delays
* Threshold for "pause" vs normal variation

See also: Inter-Token Latency (Section 4.1.3), Token Delivery Jitter (Section 4.4.3), Preemption Recovery Latency (Section 4.6.4)

4.5. Scheduling and Multi-Tenancy Metrics

4.5.1. Head-of-Line Blocking

Definition: The additional delay experienced by short requests when scheduled behind long requests in a shared queue or batch.

Discussion: First-come-first-served scheduling causes head-of-line (HOL) blocking. A long prefill operation delays all subsequent requests regardless of their size.

Chunked prefill mitigates HOL blocking by limiting the maximum uninterruptible computation. Shortest-job-first scheduling eliminates HOL blocking but requires output length prediction.

HOL blocking is measured as the difference between observed latency and latency under an idealized scheduler with no blocking.

Unit of measurement: milliseconds (ms)

Issues:

* Requires baseline comparison to quantify

* Depends on workload length distribution

See also: Prefill Latency (Section 4.2.1), Queue Wait Time (Section 4.5.3), Fairness Index (Section 4.5.4)

4.5.2. Queue Depth

Definition: The number of requests waiting for service at a measurement instant.

Discussion: Queue depth indicates load relative to capacity. Growing queue depth signals approaching overload. Stable queue depth indicates balanced load.

Queue depth has multiple measurement points: admission queue, prefill queue, decode batch wait queue. Testers MUST specify the queue measured.

Unit of measurement: requests (count)

Issues:

* Multiple queue stages in serving systems

* Instantaneous vs time-averaged measurement

See also: Offered Load (Section 4.3.5), Queue Wait Time (Section 4.5.3)

4.5.3. Queue Wait Time

Definition: The elapsed time between request arrival and scheduling for processing.

Discussion: Queue wait time is a component of TTFT representing time spent waiting rather than computing. Under low load, queue wait approaches zero. Under high load, queue wait dominates TTFT.

Systems with multiple queues (admission, prefill, decode) have corresponding wait times. Total queue wait is the sum across stages.

Unit of measurement: milliseconds (ms)

Issues:
* Multiple queue stages

* Inclusion of admission control delay

See also: Time to First Token (Section 4.1.2), Queue Depth (Section 4.5.2), Admission Rate (Section 4.5.6)

4.5.4. Fairness Index

Definition: A measure of equity in latency or throughput across concurrent requests or tenants.

Discussion: Jain’s Fairness Index quantifies allocation equality:

J(x) = (sum(x_i))^2 / (n * sum(x_i^2))

where x_i is the allocation to request or tenant i, and n is the count. J ranges from 1/n (maximally unfair) to 1 (perfectly fair).

Fairness applies to latency (lower is better, so use reciprocal), throughput, or SLO attainment. Testers MUST specify the measured quantity.

Multi-tenant systems require per-tenant fairness measurement. Single-tenant systems measure fairness across concurrent requests.

Unit of measurement: dimensionless, range [1/n, 1]

Issues:
* Choice of measured quantity (latency, throughput, SLO attainment)

* Tenant definition in multi-tenant systems

See also: Head-of-Line Blocking (Section 4.5.1), SLO Attainment Rate (Section 4.12.2)

4.5.5. Batch Utilization

Definition: The ratio of active tokens processed per batch to the maximum batch capacity.

Discussion: Static batching pads sequences to uniform length. Batch utilization measures the fraction of compute applied to real tokens versus padding.

Continuous batching achieves high utilization by avoiding padding. For continuous batching systems, batch utilization approaches 1.0 and is less informative.

Unit of measurement: ratio or percentage

Issues:
* Not meaningful for continuous batching systems

* Varies with workload length distribution

See also: Non-Padding Token Throughput (Section 4.3.4), Output Token Throughput (Section 4.3.1)

4.5.6. Admission Rate

Definition: The fraction of arriving requests accepted for processing.

Discussion: Admission control rejects requests to prevent overload. Admission rate below 1.0 indicates load shedding.

Rejected requests may receive an error response or be redirected. Testers MUST specify the rejection behavior.

Unit of measurement: ratio or percentage

Issues:

* Rejection behavior (error vs redirect)

* Distinction from content-based refusals

See also: Offered Load (Section 4.3.5), Sustainable Load (Section 4.3.6), False Refusal Rate (Section 4.11.2)

4.6. Preemption and Resource Management Metrics

4.6.1. Preemption Rate

Definition: The fraction of in-flight requests evicted from processing before completion.

Discussion: Memory pressure or priority policies cause request preemption. Preempted requests lose their key-value cache state and must recompute it upon resumption.

High preemption rates indicate memory over-subscription or aggressive scheduling. Preemption degrades latency for affected requests.

Unit of measurement: ratio or percentage

Issues:

* Distinction between temporary preemption and permanent eviction

* Treatment of requests that are preempted multiple times

See also: Preemption Loss (Section 4.6.2), Preemption Recovery Latency (Section 4.6.4), KV Cache Swap Rate (Section 4.6.5)

4.6.2. Preemption Loss

Definition: The computational work discarded when a request is preempted, measured as tokens generated before preemption that must be recomputed.

Discussion: Preemption discards key-value cache state. Upon resumption, the system recomputes prefill for all tokens (input plus previously generated output). This recomputation is wasted work.

Preemption loss contributes to tail latency. Requests preempted late in generation lose more work than those preempted early.

Unit of measurement: tokens or milliseconds of recomputation

Issues:

* Measurement requires tracking recomputed tokens

* Multiple preemptions accumulate loss

See also: Preemption Rate (Section 4.6.1), Preemption Recovery Latency (Section 4.6.4)

4.6.3. Starvation Rate

Definition: The fraction of requests waiting longer than a specified threshold before receiving any service.

Discussion: Starvation occurs when scheduling policies indefinitely delay certain requests. Priority inversion and unbounded queue growth cause starvation.

The starvation threshold depends on application requirements. Testers MUST declare the threshold when reporting starvation rate.

Unit of measurement: ratio or percentage

Issues:

* Threshold selection is application-dependent

* Distinction from queue wait time distribution tail

See also: Queue Wait Time (Section 4.5.3), Fairness Index (Section 4.5.4)

4.6.4. Preemption Recovery Latency

Definition: The elapsed time from preemption to resumption of token generation.

Discussion: Recovery latency includes: wait time for scheduling, KV cache reload or recomputation, and prefill of previously generated tokens.

Systems with KV cache offloading to host memory recover faster than those requiring full recomputation. Recovery latency varies with the amount of generated output at preemption time.

Unit of measurement: milliseconds (ms)

Issues:

* Variation based on progress at preemption

* Offloading vs recomputation recovery strategies

See also: Preemption Rate (Section 4.6.1), Preemption Loss (Section 4.6.2), KV Cache Swap Rate (Section 4.6.5)

4.6.5. KV Cache Swap Rate

Definition: The frequency at which key-value cache blocks are migrated between accelerator memory and host memory.

Discussion: Memory-constrained systems swap KV cache to host memory when accelerator memory is exhausted. Swapping enables higher concurrency at the cost of swap latency.

Swap rate indicates memory pressure. High swap rates degrade latency due to PCIe transfer overhead.

Unit of measurement: swaps per second or bytes per second

Issues:
* Granularity of swap operations (per-request vs per-block)

* Distinction between swap-out and swap-in rates

See also: Preemption Rate (Section 4.6.1), Page Fault Latency (Section 4.6.6)

4.6.6. Page Fault Latency

Definition: The latency incurred when accessing KV cache blocks not resident in accelerator memory.

Discussion: Paged attention systems fault in KV cache blocks on demand. Page fault latency includes PCIe transfer time from host memory or storage.

Fault latency increases ITL for affected decode steps. Prefetching strategies aim to hide fault latency.

Unit of measurement: milliseconds (ms)

Issues:
* Prefetching effectiveness

* Storage tier latency (DRAM vs SSD)

See also: KV Cache Swap Rate (Section 4.6.5), Inter-Token Latency (Section 4.1.3)

4.7. Prefix Caching Metrics

4.7.1. Prefix Cache Hit Rate

Definition: The fraction of input tokens whose key-value representations are retrieved from cache rather than computed.

Discussion: Prefix caching stores KV cache state for common prompt prefixes. Subsequent requests sharing a cached prefix skip prefill for those tokens.

Hit rate depends on workload locality. Workloads with shared system prompts achieve high hit rates. Workloads with unique prompts achieve low hit rates.

Hit rate is computed as cached tokens divided by total input tokens across requests. Per-request hit rate varies; report the distribution.

Unit of measurement: ratio or percentage

Issues:

* Granularity of cache matching (exact prefix vs subsequence)

* Multi-tenant cache isolation

See also: Prefill Latency (Section 4.2.1), Time to First Token (Section 4.1.2), Cache Eviction Rate (Section 4.7.3)

4.7.2. Prefix Cache Capacity

Definition: The total memory allocated for storing reusable KV cache entries.

Discussion: Cache capacity limits the number and length of prefixes stored. Larger capacity enables more prefixes or longer prefixes at the cost of memory available for active requests.

Capacity is often expressed as a fraction of total accelerator memory or as maximum cacheable token count.

Unit of measurement: bytes, tokens, or percentage of accelerator memory

Issues:

* Trade-off with memory for active requests

* Dynamic vs static allocation

See also: Prefix Cache Hit Rate (Section 4.7.1), Cache Eviction Rate (Section 4.7.3)

4.7.3. Cache Eviction Rate

Definition: The frequency at which cached prefix entries are removed to accommodate new entries.

Discussion: Eviction occurs when cache capacity is exhausted. High eviction rates indicate insufficient capacity for the workload's prefix diversity.

Eviction policies (LRU, frequency-based) affect which prefixes remain cached. Testers SHOULD specify the eviction policy.

Unit of measurement: evictions per second or evictions per request

Issues:
* Eviction policy effects
* Distinguishing capacity eviction from staleness eviction

See also: Prefix Cache Hit Rate (Section 4.7.1), Prefix Cache Capacity (Section 4.7.2)

4.7.4. TTFT Reduction from Caching

Definition: The reduction in TTFT attributable to prefix cache hits, computed as TTFT without caching minus TTFT with caching.

Discussion: This metric quantifies caching benefit. It depends on both hit rate and the length of cached prefixes.

Measurement requires comparing TTFT with caching enabled versus disabled, or estimating based on prefill latency per token.

Unit of measurement: milliseconds (ms)

Issues:
* Requires baseline measurement without caching
* Varies with cached prefix length

See also: Prefix Cache Hit Rate (Section 4.7.1), Time to First Token (Section 4.1.2), Prefill Latency (Section 4.2.1)

4.8. Speculative Decoding Metrics

4.8.1. Draft Acceptance Rate

Definition: The fraction of tokens proposed by a draft model that are accepted by the target model.

Discussion: Speculative decoding uses a smaller draft model to propose multiple tokens verified in parallel by the target model. Higher acceptance rates yield greater speedup.

Acceptance rate depends on draft model quality and alignment with the target model. Rates vary by domain; code and structured text achieve higher rates than creative writing.

Acceptance rate is computed per speculation window, then averaged. Report the distribution across windows.

Unit of measurement: ratio or percentage

Issues:

* Variation across domains and prompts
* Dependence on draft model selection

See also: Speculative Speedup (Section 4.8.2), Draft Overhead (Section 4.8.3)

4.8.2. Speculative Speedup

Definition: The ratio of decoding throughput with speculative decoding enabled to throughput with speculative decoding disabled.

Discussion: Speedup depends on acceptance rate and the relative cost of draft and target model inference. High acceptance with low draft cost yields high speedup.

Speculative decoding increases TTFT due to draft model prefill. Speedup applies to the decode phase only. End-to-end speedup is lower, especially for short outputs.

Unit of measurement: ratio (dimensionless)

Issues:

* Excludes TTFT overhead
* Varies with acceptance rate

See also: Draft Acceptance Rate (Section 4.8.1), Draft Overhead (Section 4.8.3), Output Token Throughput (Section 4.3.1)

4.8.3. Draft Overhead

Definition: The additional latency or compute cost introduced by the draft model.

Discussion: Draft model inference adds to prefill time and per-step decode time. This overhead must be recovered through acceptance to achieve net speedup.

Overhead is measured as additional latency per speculation window or as fraction of total compute.

Unit of measurement: milliseconds (ms) or percentage of total latency

Issues:

* Amortization over speculation window length
* Memory overhead for draft model weights

See also: Draft Acceptance Rate (Section 4.8.1), Speculative Speedup (Section 4.8.2)

4.8.4. Verification Throughput

Definition: The number of draft tokens verified per second by the target model.

Discussion: Verification throughput measures the target model's capacity to check draft proposals. Higher verification throughput enables longer speculation windows.

Verification processes multiple tokens in parallel, achieving higher throughput than autoregressive generation of the same tokens.

Unit of measurement: tokens per second (tok/s)

Issues:

* Distinction from generation throughput
* Variation with speculation window length

See also: Draft Acceptance Rate (Section 4.8.1), Output Token Throughput (Section 4.3.1)

4.9. Retrieval-Augmented Generation Metrics

4.9.1. Embedding Latency

Definition: The elapsed time to convert query text into vector representations for retrieval.

Discussion: Retrieval-Augmented Generation (RAG) systems embed queries before searching a vector store. Embedding latency adds to TTFT.

Embedding models are smaller and faster than generation models. Embedding latency is a minor TTFT component for most deployments.

Batched embedding of multiple queries achieves higher throughput than sequential embedding.

Unit of measurement: milliseconds (ms)

Issues:

* Batched vs sequential embedding
* Embedding model selection effects

See also: Retrieval Latency (Section 4.9.2), Time to First Token (Section 4.1.2)

4.9.2. Retrieval Latency

Definition: The elapsed time to search a vector store and fetch relevant documents.

Discussion: Retrieval latency includes vector similarity search, optional reranking, and document fetching. It is a component of TTFT for RAG systems.

Retrieval latency depends on index size, search algorithm, and number of results. Approximate nearest neighbor search trades accuracy for speed.

Unit of measurement: milliseconds (ms)

Issues:

* Index size and algorithm effects
* Reranking inclusion

See also: Embedding Latency (Section 4.9.1), Time to First Token (Section 4.1.2), Context Injection Overhead (Section 4.9.4)

4.9.3. Retrieval Recall

Definition: The fraction of relevant documents retrieved from the corpus.

Discussion: Recall measures retrieval effectiveness. Low recall causes the generation model to lack relevant context. This is a quality metric, not a performance metric, but affects overall system evaluation.

Measuring recall requires ground-truth relevance labels. For benchmarking without labels, use proxy metrics such as answer correctness.

Unit of measurement: ratio or percentage

Issues:

* Requires ground-truth relevance labels
* Trade-off with retrieval latency

See also: Retrieval Latency (Section 4.9.2), Context Utilization Rate (Section 4.9.5)

4.9.4. Context Injection Overhead

Definition: The additional prefill latency caused by retrieved context tokens.

Discussion: Retrieved documents increase prompt length, increasing prefill computation. This overhead scales with retrieved token count.

The overhead is the difference between prefill latency with retrieved context and prefill latency without it.

Unit of measurement: milliseconds (ms)

Issues:

* Varies with retrieval result length
* Interaction with context length limits

See also: Prefill Latency (Section 4.2.1), Retrieval Latency (Section 4.9.2)

4.9.5. Context Utilization Rate

Definition: The fraction of retrieved context tokens that materially influence the generated response.

Discussion: Not all retrieved content contributes to generation. Low utilization indicates retrieval of irrelevant content, wasting context window capacity and prefill compute.

Utilization is difficult to measure directly. Proxy measurements include attention weight analysis and ablation studies.

Unit of measurement: ratio or percentage

Issues:

* Indirect measurement required
* Attribution challenges

See also: Retrieval Recall (Section 4.9.3), Context Injection Overhead (Section 4.9.4)

4.10. Agentic and Compound System Metrics

4.10.1. Task Completion Latency

Definition: The elapsed time for a compound system to satisfy a user intent, encompassing all LLM calls, retrieval steps, and tool executions.

Discussion: Agentic systems perform multiple internal operations per user request. Task completion latency measures the full user-facing response time.

Task completion latency depends on the number of internal steps, which varies by task complexity. Simple tasks complete in one LLM call. Complex tasks require multiple calls with tool use.

Unit of measurement: seconds (s)

Issues:

* Variation with task complexity
* Definition of task completion for open-ended tasks

See also: Sub-Request Count (Section 4.10.2), Tool Execution Latency (Section 4.10.4)

4.10.2. Sub-Request Count

Definition: The number of internal LLM inference requests triggered by a single user interaction.

Discussion: Agentic systems decompose user requests into multiple LLM calls for planning, reasoning, and action. Sub-request count indicates system complexity and affects total latency and cost.

High sub-request counts indicate complex reasoning chains or retry loops. Testers SHOULD examine sub-request count distributions to identify inefficient patterns.

Unit of measurement: count

Issues:

* Includes retries and failed attempts
* Varies with task complexity

See also: Task Completion Latency (Section 4.10.1), Loop Incidence Rate (Section 4.10.3)

4.10.3. Loop Incidence Rate

Definition: The fraction of tasks where the agent enters a repetitive control flow without making progress.

Discussion: Agentic loops occur when the system repeats similar actions without advancing toward task completion. Loops indicate planning failures or tool errors.

Loop detection requires defining "progress" for the task domain. Common heuristics include action repetition count and state similarity thresholds.

Unit of measurement: ratio or percentage

Issues:

* Progress definition is task-dependent
* Distinguishing loops from legitimate retries

See also: Sub-Request Count (Section 4.10.2), Task Completion Latency (Section 4.10.1)

4.10.4. Tool Execution Latency

Definition: The elapsed time for external tool calls invoked by the agent.

Discussion: Agents call external tools for information retrieval, computation, or actions. Tool latency contributes to task completion latency.

Tool latency varies by tool type. Local computation completes in milliseconds. External API calls require seconds.

Unit of measurement: milliseconds (ms) or seconds (s)

Issues:

* High variance across tool types
* External dependencies outside system control

See also: Task Completion Latency (Section 4.10.1), Sub-Request Count (Section 4.10.2)

4.10.5. Agentic Goodput

Definition: The fraction of tasks completed successfully while meeting declared task-level objectives.

Discussion: Agentic goodput combines completion rate and quality. A task that completes but produces incorrect results does not count toward goodput.

Objective definitions are task-specific. Testers MUST declare objectives and evaluation criteria.

Unit of measurement: ratio or percentage

Issues:

* Task-specific objective definitions
* Evaluation criteria for correctness

See also: Task Completion Latency (Section 4.10.1), SLO Attainment Rate (Section 4.12.2)

4.11. Quality and Policy Enforcement Metrics

4.11.1. Policy Violation Rate

Definition: The fraction of responses that violate declared content or safety policies.

Discussion: Safety systems filter outputs to prevent harmful content. Policy violation rate measures filter failure.

Violation detection requires evaluation against policy criteria. Automated classifiers or human review provide measurements.

Low violation rate indicates effective filtering. Very low rates may indicate overly restrictive filtering causing false refusals.

Unit of measurement: ratio or percentage

Issues:

* Policy definition and scope
* Detection method reliability

See also: False Refusal Rate (Section 4.11.2), Guardrail Processing Overhead (Section 4.11.3)

4.11.2. False Refusal Rate

Definition: The fraction of benign, policy-compliant requests that are incorrectly refused.

Discussion: Overly sensitive safety filters refuse legitimate requests. False refusals degrade user experience and system utility.

Measuring false refusals requires labeled benign requests. Testers MUST specify the evaluation dataset and labeling criteria.

False refusal rate trades off against policy violation rate. Stricter filtering reduces violations but increases false refusals.

Unit of measurement: ratio or percentage

Issues:

* Requires labeled benign test set
* Trade-off with policy violation rate

See also: Policy Violation Rate (Section 4.11.1), Admission Rate

(Section 4.5.6)

4.11.3. Guardrail Processing Overhead

Definition: The additional latency introduced by safety, policy, or content filtering mechanisms.

Discussion: Guardrails add processing before, during, or after generation. Input filters add to TTFT. Output filters add to end-to-end latency.

Overhead is measured by comparing latency with guardrails enabled versus disabled.

Unit of measurement: milliseconds (ms)

Issues:

* Baseline measurement without guardrails
* Multiple guardrail stages with distinct overhead

See also: Time to First Token (Section 4.1.2), End-to-End Latency (Section 4.1.1), Policy Violation Rate (Section 4.11.1)

4.12. Service Level Objective Metrics

4.12.1. Service Level Objective

Definition: A quantitative threshold for a performance metric that defines acceptable service quality.

Discussion: SLOs specify targets such as:

* P99 TTFT below 500ms
* P95 TPOT below 50ms
* Error rate below 0.1%

SLOs derive from user experience requirements and business constraints. Different applications require different SLOs.

SLO definitions include the metric, percentile, threshold, and measurement window. Testers MUST fully specify SLOs when reporting attainment.

Unit of measurement: not applicable (SLO is a specification, not a measurement)

Issues:

* Application-specific requirements

* Multiple SLOs may apply simultaneously

See also: SLO Attainment Rate (Section 4.12.2), Sustainable Load (Section 4.3.6)

4.12.2. SLO Attainment Rate

Definition: The fraction of requests or time periods meeting all declared SLOs.

Discussion: Attainment rate summarizes SLO compliance. Request-based attainment counts requests meeting SLOs. Time-based attainment counts measurement windows where aggregate metrics meet SLOs.

Attainment below 1.0 indicates SLO violations. The acceptable attainment level depends on SLA commitments.

Unit of measurement: ratio or percentage

Issues:

* Request-based vs time-based measurement

* Handling of multiple simultaneous SLOs

See also: Service Level Objective (Section 4.12.1), Sustainable Load (Section 4.3.6)

4.12.3. Goodput

Definition: The throughput of requests that complete successfully and meet SLO requirements.

Discussion: Goodput equals total throughput multiplied by SLO attainment rate. It measures useful, compliant work rather than raw capacity.

Systems with high throughput but low attainment achieve low goodput. Goodput captures the throughput-quality trade-off.

Unit of measurement: requests per second (req/s) or tokens per second (tok/s)

Issues:

* Requires defined SLOs

* May use request or token basis

See also: SLO Attainment Rate (Section 4.12.2), Output Token Throughput (Section 4.3.1), Request Throughput (Section 4.3.3)

5. Measurement Considerations

5.1. Workload Specification

Benchmarks MUST specify the workload used for measurement:

* Input length distribution (mean, percentiles, min, max)

* Output length distribution or generation parameters

* Request arrival pattern (open-loop rate or closed-loop concurrency)

* Dataset source or generation method

* Tokenizer used for length calculations

Workload characteristics affect all metrics. Results from different workloads are not directly comparable.

5.2. Warm-up and Steady State

Systems require warm-up before reaching steady-state performance. Warm-up effects include:

* JIT compilation of kernels

* Memory allocator warm-up

* Prefix cache population

* Batch size ramp-up

Testers MUST exclude warm-up from measurement or report it separately. Testers SHOULD document the warm-up procedure and duration.

5.3. Measurement Duration

Measurement windows MUST be long enough to capture steady-state behavior and sufficient samples for statistical reliability.

For percentile measurements:

* P50 requires at least 100 samples

* P99 requires at least 1,000 samples

* P99.9 requires at least 10,000 samples

Testers MUST report sample counts alongside percentiles.

5.4. Clock Synchronization

Distributed systems require synchronized clocks for latency measurement. Clock skew introduces measurement error.

Testers SHOULD use NTP or PTP for clock synchronization and report the synchronization method and estimated accuracy.

5.5. System Configuration

Benchmarks MUST report system configuration:

* Hardware: accelerator model, count, memory, interconnect

* Software: serving framework, version, model format

* Model: architecture, parameter count, precision

* Serving: batching strategy, parallelism configuration

* Tuning: any non-default configuration parameters

Results are specific to the reported configuration.

6. Security Considerations

6.1. Threat Model

This section considers adversaries who submit requests designed to degrade service for other users or extract information about the system or other users' requests.

Performance benchmarking itself does not introduce security vulnerabilities. However, performance characteristics may be exploited by adversaries.

6.2. Side-Channel Information Leakage

Shared infrastructure creates side-channel risks.

Timing channels: Request latency depends on queue depth, batch composition, and cache state influenced by other users. An adversary observing their own request latency may infer information about concurrent requests.

Cache channels: Prefix caching creates observable timing differences between cache hits and misses. An adversary may probe for cached prefixes to learn about other users' prompts.

Batch channels: Continuous batching causes ITL variation based on batch membership changes. An adversary may infer when other requests arrive or complete.

Mitigation strategies include request isolation, timing noise injection, and partitioned caching. These mitigations affect performance. Testers evaluating multi-tenant systems SHOULD measure side-channel leakage alongside performance.

6.3. Resource Exhaustion Attacks

Adversaries may craft requests to exhaust system resources:

Memory exhaustion: Requests with long outputs grow KV cache until memory is exhausted. Systems without output length limits or memory management are vulnerable.

Compute exhaustion: Long input sequences maximize prefill compute. Pathological inputs may trigger worst-case attention patterns.

Queue exhaustion: Bursts of requests exceed admission capacity. Without rate limiting, legitimate requests are delayed or rejected.

The metrics Sustainable Load (Section 4.3.6), and Admission Rate (Section 4.5.6) characterize resilience to resource exhaustion.

6.4. Model Extraction

High-volume query access enables model extraction attacks where an adversary trains a copy of the model from input-output pairs. This document does not define rate-limiting terminology. Deployments concerned with model extraction SHOULD implement and monitor rate limits.

6.5. Benchmark Gaming

Systems may be optimized for benchmark workloads in ways that do not generalize to production traffic. Testers SHOULD use diverse workloads representative of intended deployment.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[RFC1242] Bradner, S., "Benchmarking Terminology for Network Interconnection Devices", RFC 1242, DOI 10.17487/RFC1242, July 1991, <https://www.rfc-editor.org/info/rfc1242>.

[RFC2544] Bradner, S. and J. McQuaid, "Benchmarking Methodology for Network Interconnect Devices", RFC 2544, DOI 10.17487/RFC2544, March 1999, <https://www.rfc-editor.org/info/rfc2544>.

[RFC2647] Newman, D., "Benchmarking Terminology for Firewall Performance", RFC 2647, DOI 10.17487/RFC2647, August 1999, <https://www.rfc-editor.org/info/rfc2647>.

[MLPERF] Reddi, V. J., "MLPerf Inference Benchmark", Proceedings of the ACM/IEEE International Symposium on Computer Architecture (ISCA), DOI 10.1109/ISCA45697.2020.00045, 2020, <https://doi.org/10.1109/ISCA45697.2020.00045>.

[VLLM] Kwon, W., "Efficient Memory Management for Large Language Model Serving with PagedAttention", Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), DOI 10.1145/3600006.3613165, 2023, <https://doi.org/10.1145/3600006.3613165>.

[ORCA] Yu, G., "Orca: A Distributed Serving System for Transformer-Based Generative Models", Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2022.

[SARATHI] Agrawal, A., "Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve", Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2024.

[ATTENTION]
Vaswani, A., "Attention Is All You Need", Advances in Neural Information Processing Systems (NeurIPS), 2017.

[JAIN] Jain, R., Chiu, D., and W. Hawe, "A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer Systems", DEC Research Report TR-301, September 1984.

Appendix A. Supplementary Metrics

This appendix defines metrics relevant for production deployment decisions but not essential for basic performance characterization.

A.1. Energy and Sustainability Metrics

A.1.1. Energy per Token

Definition: The energy consumed to generate one output token.

Discussion: Energy per token enables efficiency comparison across systems and sustainability analysis. The value depends on model size, hardware, batch size, and workload.

Measurement requires power monitoring integrated with token counting. GPU power is accessible via vendor APIs (NVML for NVIDIA). Total system power requires external instrumentation.

Energy per token differs between prefill and decode phases. Testers SHOULD report phase-separated energy when feasible.

Unit of measurement: Joules per token (J/tok) or millijoules per token (mJ/tok)

Issues:
* Measurement boundary (GPU vs system)

* Phase attribution

See also: Tokens per Joule (Appendix A.1.2), Output Token Throughput (Section 4.3.1)

A.1.2. Tokens per Joule

Definition: The number of output tokens generated per Joule of energy.

Discussion: Tokens per Joule is the inverse of energy per token,

expressing energy efficiency. Higher values indicate greater efficiency.

Unit of measurement: tokens per Joule (tok/J)

Issues: Same as Energy per Token

See also: Energy per Token (Appendix A.1.1)

A.1.3. Instantaneous Power Draw

Definition: The electrical power consumed at a given instant.

Discussion: Power draw varies with load. Idle systems consume less power than systems under load. Prefill phases consume more power than decode phases due to higher compute utilization.

Testers MUST specify the measurement boundary: GPU only, GPU and CPU, or entire system including cooling.

Unit of measurement: Watts (W)

Issues:

* Measurement boundary specification
* Sampling rate for time-varying power

See also: Peak Power Draw (Appendix A.1.4), Idle Power Draw (Appendix A.1.5)

A.1.4. Peak Power Draw

Definition: The maximum instantaneous power observed during a measurement interval.

Discussion: Peak power determines infrastructure requirements for power delivery and cooling. Systems may have brief power spikes exceeding average consumption.

Unit of measurement: Watts (W)

Issues:

* Sampling rate may miss brief spikes
* Relationship to thermal design power (TDP)

See also: Instantaneous Power Draw (Appendix A.1.3)

A.1.5. Idle Power Draw

Definition: The power consumed when the system is ready to serve but not processing requests.

Discussion: Idle power represents the baseline cost of provisioned capacity. The difference between loaded and idle power indicates dynamic power range.

Unit of measurement: Watts (W)

Issues: Definition of idle (model loaded, empty batch)

See also: Instantaneous Power Draw (Appendix A.1.3)

A.1.6. Carbon Intensity

Definition: The greenhouse gas emissions per token or per request.

Discussion: Carbon intensity equals energy consumption multiplied by the grid's carbon factor (gCO2e/kWh). Grid carbon intensity varies by location and time.

Testers reporting carbon metrics MUST specify the grid carbon factor used.

Unit of measurement: grams CO2 equivalent per token (gCO2e/tok)

Issues:

* Grid carbon factor variation

* Scope of emissions (operational only or including embodied)

See also: Energy per Token (Appendix A.1.1)

A.2. Economic and Cost Metrics

A.2.1. Cost per Million Tokens

Definition: The monetary cost to generate one million output tokens.

Discussion: Cost includes compute (hardware amortization or rental), energy, and operations. Testers MUST specify included cost components.

Cloud pricing provides a market cost reference. Self-hosted deployments require cost modeling.

Unit of measurement: currency per million tokens (e.g., $/Mtok)

Issues:
* Cost component inclusion

* Pricing model assumptions

See also: GPU-Hours per Request (Appendix A.2.2), Throughput-Cost Ratio (Appendix A.2.3)

A.2.2. GPU-Hours per Request

Definition: The accelerator time consumed to complete one request.

Discussion: GPU-hours measures resource consumption independent of hardware cost. For multi-GPU deployments, report aggregate GPU-hours.

GPU-hours equals end-to-end latency multiplied by GPU count used for the request.

Unit of measurement: GPU-hours

Issues:
* Fractional allocation in multi-tenant systems

* GPU count for distributed inference

See also: End-to-End Latency (Section 4.1.1), Cost per Million Tokens (Appendix A.2.1)

A.2.3. Throughput-Cost Ratio

Definition: The ratio of sustainable throughput to infrastructure cost.

Discussion: Higher ratios indicate better cost efficiency. Testers MUST specify the throughput metric (tokens or requests) and cost basis (hourly rental or amortized ownership).

Unit of measurement: tokens per second per dollar-hour (tok/s/$h)

Issues:
* Cost basis selection

* Throughput metric selection

See also: Sustainable Load (Section 4.3.6), Cost per Million Tokens

(Appendix A.2.1)

A.3. Hardware Utilization Metrics

A.3.1. Compute Utilization

Definition: The fraction of available compute capacity in use.

Discussion: Compute utilization indicates how effectively the system uses accelerator compute resources. Low utilization suggests memory bandwidth or scheduling bottlenecks.

For GPUs, utilization metrics include SM occupancy and tensor core utilization. Testers MUST specify the utilization metric.

Unit of measurement: percentage

Issues:

* Multiple utilization definitions (SM, tensor core, etc.)

* Vendor-specific measurement tools

See also: Memory Bandwidth Utilization (Appendix A.3.2)

A.3.2. Memory Bandwidth Utilization

Definition: The fraction of peak memory bandwidth consumed.

Discussion: LLM decode is memory-bandwidth-bound on current hardware. High bandwidth utilization indicates the system is limited by memory throughput rather than compute.

Unit of measurement: percentage

Issues:

* Measurement tool availability

* Cache effects on apparent bandwidth

See also: Compute Utilization (Appendix A.3.1)

A.3.3. KV Cache Memory

Definition: The memory consumed by key-value cache for active requests.

Discussion: KV cache memory grows with batch size and sequence length. Memory exhaustion limits concurrent request capacity.

KV cache memory equals: batch_size * sequence_length * num_layers * 2 * hidden_dim * precision_bytes

Unit of measurement: gigabytes (GB) or percentage of accelerator memory

Issues:
* Variation with batch composition

* Paged vs contiguous allocation

See also: KV Cache Swap Rate (Section 4.6.5), Page Fault Latency (Section 4.6.6)

A.3.4. Memory Fragmentation

Definition: The fraction of free memory unusable for new allocations due to non-contiguous layout.

Discussion: Memory fragmentation reduces effective capacity. Paged attention systems achieve low fragmentation through fine-grained allocation.

Unit of measurement: percentage

Issues:
* Allocator-specific measurement

* Definition of "unusable" depends on minimum allocation size

See also: KV Cache Memory (Appendix A.3.3)

A.4. Distributed Serving Metrics

A.4.1. Tensor Parallel Efficiency

Definition: The ratio of achieved throughput to ideal linear scaling with tensor parallelism degree.

Discussion: Tensor parallelism partitions model layers across accelerators. Communication overhead reduces efficiency below ideal scaling.

Efficiency equals throughput with N GPUs divided by (N times single-GPU throughput).

Unit of measurement: percentage

Issues:

* Baseline single-GPU measurement

* Communication topology effects

See also: Pipeline Parallel Efficiency (Appendix A.4.2), Collective Communication Latency (Appendix A.4.4)

A.4.2. Pipeline Parallel Efficiency

Definition: The ratio of achieved throughput to ideal linear scaling with pipeline parallelism depth.

Discussion: Pipeline parallelism partitions model layers into sequential stages. Pipeline bubbles (idle time during fill and drain) reduce efficiency.

Bubble fraction equals (P-1)/(P-1+M) where P is pipeline depth and M is microbatch count.

Unit of measurement: percentage

Issues:

* Microbatch count selection

* Stage imbalance effects

See also: Tensor Parallel Efficiency (Appendix A.4.1)

A.4.3. Expert Parallel Load Imbalance

Definition: The deviation of token routing from uniform distribution across experts in Mixture-of-Experts models.

Discussion: Load imbalance causes some experts to become bottlenecks while others are underutilized. Imbalance metrics include coefficient of variation or max/mean ratio.

Unit of measurement: dimensionless ratio or percentage

Issues:

* Imbalance metric selection

* Dynamic variation across batches

See also: Tensor Parallel Efficiency (Appendix A.4.1)

A.4.4. Collective Communication Latency

Definition: The latency of collective operations (all-reduce, all-gather) used in distributed inference.

Discussion: Collective communication occurs between forward pass segments in tensor-parallel inference. This latency adds to per-token generation time.

Unit of measurement: microseconds (us) or milliseconds (ms)

Issues:
* Operation type specification
* Message size effects

See also: Tensor Parallel Efficiency (Appendix A.4.1), Inter-Token Latency (Section 4.1.3)

A.5. Quantization and Precision Metrics

A.5.1. Precision Mode

Definition: The numerical representation for model weights and activations.

Discussion: Common precision modes include FP32, FP16, BF16, FP8, INT8, and INT4. Lower precision reduces memory and increases throughput at potential accuracy cost.

Mixed precision uses different precisions for different tensors. Testers MUST specify precision for weights, activations, and KV cache separately if they differ.

Unit of measurement: not applicable (categorical specification)

Issues:
* Mixed precision specification
* KV cache precision separate from weights

See also: Quantization Speedup (Appendix A.5.2), Quantization Accuracy Impact (Appendix A.5.4)

A.5.2. Quantization Speedup

Definition: The throughput improvement from quantization relative to a baseline precision.

Discussion: Speedup equals quantized throughput divided by baseline throughput. Testers MUST specify the baseline precision.

Unit of measurement: dimensionless ratio

Issues:
* Baseline precision selection
* Workload effects on speedup

See also: Precision Mode (Appendix A.5.1), Quantization Memory Reduction (Appendix A.5.3)

A.5.3. Quantization Memory Reduction

Definition: The reduction in model memory from quantization relative to a baseline precision.

Discussion: Memory reduction enables larger batch sizes or longer sequences. Reduction equals 1 minus (quantized size divided by baseline size).

Unit of measurement: percentage

Issues:
* Baseline precision selection
* Overhead from quantization metadata

See also: Precision Mode (Appendix A.5.1), KV Cache Memory (Appendix A.3.3)

A.5.4. Quantization Accuracy Impact

Definition: The change in model quality metrics due to quantization.

Discussion: Quantization may degrade accuracy. Impact is measured on task- specific benchmarks or perplexity.

Testers MUST specify the evaluation benchmark and baseline.

Unit of measurement: percentage points or absolute score change

Issues:
* Benchmark selection
* Task-specific variation

See also: Precision Mode (Appendix A.5.1)

A.6. Operational Lifecycle Metrics

A.6.1. Model Load Time

Definition: The elapsed time from process start to model readiness for serving.

Discussion: Load time includes weight loading from storage, memory allocation, and initialization. Load time affects scale-up responsiveness in autoscaling deployments.

Unit of measurement: seconds (s)

Issues:
* Storage medium effects (local vs network)
* Parallelism in loading

See also: Cold Start Latency (Appendix A.6.2), Scale-Up Latency (Appendix A.6.4)

A.6.2. Cold Start Latency

Definition: The latency of the first request after system initialization.

Discussion: Cold start latency exceeds steady-state latency due to JIT compilation, cache population, and memory allocation.

Cold start affects user experience for scale-to-zero deployments and after restarts.

Unit of measurement: milliseconds (ms)

Issues:
* Definition of cold (first request vs first after idle)
* JIT compilation effects

See also: Model Load Time (Appendix A.6.1), Warm-up Duration (Appendix A.6.3)

A.6.3. Warm-up Duration

Definition: The time or request count until latency stabilizes at steady state.

Discussion: Warm-up accounts for JIT compilation, cache warming, and allocator stabilization. Testers SHOULD exclude warm-up from steady-state measurements.

Unit of measurement: seconds (s) or request count

Issues:
* Stability definition
* Workload effects on warm-up

See also: Cold Start Latency (Appendix A.6.2)

A.6.4. Scale-Up Latency

Definition: The elapsed time to add serving capacity and begin handling traffic.

Discussion: Scale-up latency affects autoscaling responsiveness. It includes instance provisioning, model loading, and warm-up.

Unit of measurement: seconds (s)

Issues:
* Infrastructure-dependent (containers, VMs, bare metal)
* Warm traffic routing timing

See also: Model Load Time (Appendix A.6.1), Cold Start Latency (Appendix A.6.2)

A.7. Long-Context Metrics

A.7.1. Maximum Context Length

Definition: The maximum combined input and output token count supported by the system.

Discussion: Context length is limited by model architecture, memory capacity, and serving configuration. Systems may support longer contexts than advertised at degraded performance.

Unit of measurement: tokens

Issues:
* Architectural vs memory-constrained limits
* Performance degradation at maximum length

See also: Context Window Utilization (Appendix A.7.2)

A.7.2. Context Window Utilization

Definition: The fraction of maximum context length consumed by a request.

Discussion: Utilization equals (input tokens plus output tokens) divided by maximum context length. High utilization stresses memory and may degrade performance.

Unit of measurement: percentage

Issues:
* Maximum length definition
* Performance effects at high utilization

See also: Maximum Context Length (Appendix A.7.1)

A.7.3. Long-Context Latency Scaling

Definition: The rate at which latency increases with context length.

Discussion: Prefill latency scales linearly with input length for standard attention. Decode latency scales with total sequence length due to growing KV cache access.

Efficient attention mechanisms (sparse, linear) may achieve sub-linear scaling.

Unit of measurement: milliseconds per token (ms/tok) or scaling exponent

Issues:
* Prefill vs decode scaling differs
* Architecture-dependent scaling

See also: Prefill Latency (Section 4.2.1), Decode Latency (Section 4.2.2)

Appendix B. Cross-Reference Index

This index groups metrics by relationship for navigation.

Latency metrics:

* End-to-End Latency (Section 4.1.1)
* Time to First Token (Section 4.1.2)
* Inter-Token Latency (Section 4.1.3)
* Time Between Tokens (Section 4.1.4)
* Time per Output Token (Section 4.1.5)
* Normalized Latency (Section 4.1.6)
* Prefill Latency (Section 4.2.1)
* Decode Latency (Section 4.2.2)

Throughput metrics:

* Output Token Throughput (Section 4.3.1)
* Input Token Throughput (Section 4.3.2)
* Request Throughput (Section 4.3.3)
* Non-Padding Token Throughput (Section 4.3.4)
* Goodput (Section 4.12.3)

Distribution metrics:

* Latency Percentiles (Section 4.4.1)
* Latency Distribution (Section 4.4.2)
* Token Delivery Jitter (Section 4.4.3)
* Maximum Pause Duration (Section 4.4.4)

Scheduling metrics:

* Head-of-Line Blocking (Section 4.5.1)
* Queue Depth (Section 4.5.2)
* Queue Wait Time (Section 4.5.3)
* Fairness Index (Section 4.5.4)

* Batch Utilization (Section 4.5.5)
* Admission Rate (Section 4.5.6)

Resource management metrics:

* Preemption Rate (Section 4.6.1)
* Preemption Loss (Section 4.6.2)
* Starvation Rate (Section 4.6.3)
* Preemption Recovery Latency (Section 4.6.4)
* KV Cache Swap Rate (Section 4.6.5)
* Page Fault Latency (Section 4.6.6)

Caching metrics:

* Prefix Cache Hit Rate (Section 4.7.1)
* Prefix Cache Capacity (Section 4.7.2)
* Cache Eviction Rate (Section 4.7.3)
* TTFT Reduction from Caching (Section 4.7.4)

Speculative decoding metrics:

* Draft Acceptance Rate (Section 4.8.1)
* Speculative Speedup (Section 4.8.2)
* Draft Overhead (Section 4.8.3)
* Verification Throughput (Section 4.8.4)

RAG metrics:

* Embedding Latency (Section 4.9.1)
* Retrieval Latency (Section 4.9.2)
* Retrieval Recall (Section 4.9.3)
* Context Injection Overhead (Section 4.9.4)

* Context Utilization Rate (Section 4.9.5)

Agentic metrics:

* Task Completion Latency (Section 4.10.1)
* Sub-Request Count (Section 4.10.2)
* Loop Incidence Rate (Section 4.10.3)
* Tool Execution Latency (Section 4.10.4)
* Agentic Goodput (Section 4.10.5)

Quality metrics:

* Policy Violation Rate (Section 4.11.1)
* False Refusal Rate (Section 4.11.2)
* Guardrail Processing Overhead (Section 4.11.3)

SLO metrics:

* Service Level Objective (Section 4.12.1)
* SLO Attainment Rate (Section 4.12.2)
* Goodput (Section 4.12.3)

Author's Address

Madhava Gaikwad
Independent Researcher
Email: gaikwad.madhav@gmail.com