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

M. Gaikwad
Independent Researcher

January 2026

Benchmarking Methodology for Large Language Model Serving
draft-gaikwad-llm-benchmarking-methodology-00

Abstract

This document defines benchmarking methodologies for Large Language Model (LLM) inference serving systems. It provides test procedures, setup parameters, measurement specifications, and reporting formats for evaluating latency, throughput, scheduling, and resource management characteristics. This document is a companion to "Benchmarking Terminology for Large Language Model Serving" and SHOULD be consulted alongside that terminology document.

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

This document provides benchmarking methodologies for Large Language Model inference serving systems. It defines test procedures, measurement specifications, and reporting formats that enable meaningful performance comparison.

A companion document, "Benchmarking Terminology for Large Language Model Serving" [LLM-TERMS], defines the metrics referenced in this methodology. That terminology document SHOULD be consulted before attempting to make use of this document.

LLM serving systems present unique benchmarking challenges:

Streaming responses: Output tokens arrive incrementally over seconds or minutes, requiring timing measurements at multiple points within a single request.

Phase separation: The prefill phase (processing input) and decode phase (generating output) have distinct computational profiles and optimization targets.

Memory-bound decoding: The decode phase is limited by memory bandwidth rather than compute, creating different bottlenecks than traditional neural network inference.

Dynamic batching: Continuous batching systems interleave requests, causing per-request performance to depend on concurrent load.

Context-dependent performance: Request latency varies with input length, output length, and cache state, making workload specification critical.

These characteristics require methodology beyond traditional throughput and latency measurement. This document addresses these challenges by specifying:

* Test configurations for different system boundaries

* Reference workloads with defined characteristics

* Measurement procedures for streaming responses

* Statistical requirements for reliable percentile estimation

* Reporting formats enabling meaningful comparison

This document does not specify acceptance thresholds or recommend particular systems. It provides methodology for fair comparison.

2. 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.

An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for a given test. An implementation that satisfies all the MUST and all the SHOULD requirements for a test is said to be "unconditionally compliant" for that test; one that satisfies all the MUST requirements but not all the SHOULD requirements is said to be "conditionally compliant."

3. Scope

This document covers benchmarking methodology for transformer-based autoregressive language models deployed as network services. The methodology applies to:

* Inference engines executing model forward passes

* Application gateways providing API endpoints

* Compound systems with retrieval or tool execution

The following are out of scope:

* Model training or fine-tuning performance

* Model quality or accuracy evaluation

* Non-autoregressive models (diffusion, encoder-only)

* Edge deployment or on-device inference

* Specific vendor implementations or products

4. Test Setup

4.1. System Under Test Configurations

The System Under Test (SUT) boundary MUST be declared before benchmarking. This document defines three standard configurations.

4.1.1. Model Engine Configuration

The Model Engine configuration measures raw inference capability.

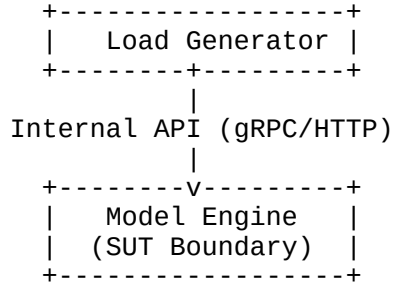

Figure 1: Model Engine Configuration

Included components:

* Model weights and inference runtime

* Batching and scheduling logic

* KV cache management

* Tensor operations and kernels

Excluded components:

* External network transport

* Authentication and authorization

* Rate limiting

* Input/output safety filtering

* Load balancing

This configuration is appropriate for comparing inference engines (vLLM, TensorRT-LLM, SGLang) independent of deployment stack.

4.1.2. Application Gateway Configuration

The Application Gateway configuration measures user-observable API performance.

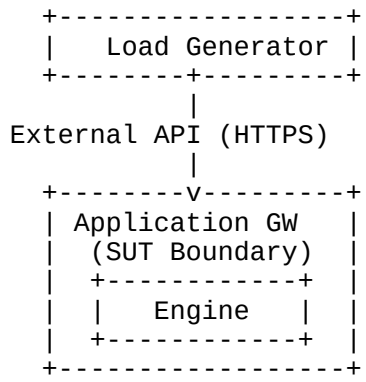

Figure 2: Application Gateway Configuration

Included components (in addition to Model Engine):

* TLS termination

* Authentication and session management

* Rate limiting and quota enforcement

* Input validation and output filtering

* Safety guardrails

This configuration is appropriate for comparing API providers or evaluating production deployment performance.

4.1.3. Compound System Configuration

The Compound System configuration measures end-to-end task completion for agentic or retrieval-augmented workloads.

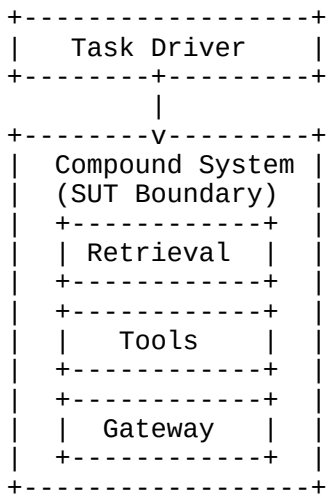

Figure 3: Compound System Configuration

Included components (in addition to Application Gateway):

* Retrieval pipeline (embedding, vector search, reranking)
* Tool execution environment
* Orchestration logic
* Multi-turn conversation state

This configuration is appropriate for evaluating RAG systems or agentic applications.

4.2. Load Generator Requirements

The load generator produces requests and measures responses. It MUST satisfy the following requirements.

4.2.1. Timing Resolution

The load generator MUST measure time with resolution of 1 millisecond or better. Microsecond resolution is RECOMMENDED for ITL measurement.

4.2.2. Streaming Support

The load generator MUST support streaming response protocols (SSE, WebSocket, or gRPC streaming). It MUST record the arrival time of each token or chunk, not only the complete response.

4.2.3. Open-Loop Load Generation

The load generator MUST support open-loop load generation where request arrival times are determined by a specified distribution independent of response times. Poisson arrivals MUST be supported. Uniform and bursty arrival patterns are RECOMMENDED.

4.2.4. Closed-Loop Load Generation

The load generator MUST support closed-loop load generation where a fixed number of concurrent requests are maintained. When a request completes, a new request is immediately submitted.

4.2.5. Request Isolation

The load generator MUST NOT allow slow responses to delay the submission of subsequent requests in open-loop mode. Asynchronous or multi-threaded implementation is REQUIRED.

4.2.6. Output Recording

The load generator MUST record for each request:

* Request submission timestamp
* First token arrival timestamp
* Each subsequent token arrival timestamp
* Final token arrival timestamp
* Total input token count
* Total output token count
* Request success/failure status

4.3. Reference Workloads

Workload specification is critical for reproducible benchmarking. This document defines reference workloads with fixed characteristics. Testers MAY use custom workloads but MUST fully specify them.

4.3.1. Workload Parameters

Each workload MUST specify:

Input length distribution: Distribution type (fixed, uniform, normal, empirical), parameters (mean, std, min, max, or histogram), and unit (tokens using specified tokenizer).

Output length distribution: Distribution type (fixed, uniform, normal, empirical), parameters (mean, std, min, max, or histogram), control method (max_tokens parameter, stop sequence, or both), and unit (tokens using specified tokenizer).

Content characteristics: Domain (general, code, conversation, instruction), language (English, multilingual, code languages), and system prompt presence and typical length.

Prefix sharing: Fraction of requests sharing common prefix and shared prefix length distribution.

4.3.2. Standard Workloads

This document defines five standard workloads. Full specifications appear in Appendix A.

4.3.2.1. Synthetic-Uniform

Purpose: Baseline comparison with controlled variability

* Input length: Uniform(128, 512) tokens
* Output length: Uniform(64, 256) tokens
* Content: Random token sequences (no semantic meaning)
* Prefix sharing: None

This workload isolates inference performance from content effects. It is REQUIRED for Model Engine benchmarking.

4.3.2.2. Synthetic-Skewed

Purpose: Test behavior under realistic length variation

* Input length: Log-normal(mu=5.5, sigma=1.0) tokens, capped at 4096
* Output length: Log-normal(mu=4.5, sigma=1.2) tokens, capped at 2048

* Content: Random token sequences
* Prefix sharing: None

This workload tests scheduling fairness with high length variance.

4.3.2.3. Conversation

Purpose: Simulate interactive chat workloads

* Input length: Empirical distribution from ShareGPT dataset
* Output length: Empirical distribution from ShareGPT dataset
* Content: Natural language conversation
* Prefix sharing: 50% share 200-token system prompt

This workload is RECOMMENDED for Application Gateway benchmarking.

4.3.2.4. Code Completion

Purpose: Simulate coding assistant workloads

* Input length: Empirical from code completion datasets
* Output length: Log-normal(mu=4.0, sigma=1.5) tokens
* Content: Source code in Python, JavaScript, TypeScript
* Prefix sharing: 80% share repository context prefix

This workload tests prefix caching effectiveness.

4.3.2.5. Long Context

Purpose: Test long-context behavior

* Input length: Uniform(8192, 32768) tokens
* Output length: Fixed at 256 tokens
* Content: Document + question format
* Prefix sharing: None

This workload is REQUIRED for Long Context Scaling tests.

4.3.3. Workload Reproducibility

For reproducible benchmarking:

* Testers MUST use deterministic random seeds for workload generation. The seed MUST be reported.

* Testers SHOULD publish the exact request sequences used, or provide generation code with fixed seeds.

* When using dataset-derived workloads (ShareGPT, HumanEval), testers MUST specify the dataset version, subset selection method, and any preprocessing applied.

4.4. Tokenization

Token counts depend on the tokenizer. Different tokenizers produce different counts for identical text, making cross-system comparison challenging.

4.4.1. Tokenizer Specification

The test report MUST specify:

* Tokenizer name and version (e.g., "cl100k_base", "Llama-3 tokenizer")

* Vocabulary size

* Source (Hugging Face model ID, tiktoken name, or custom)

4.4.2. Token Counting Method

For cross-system comparison where systems use different tokenizers:

Option A - Native tokenizer: Count tokens using each system's native tokenizer. Report results separately with tokenizer identified. This method reflects actual system behavior but complicates comparison.

Option B - Reference tokenizer: Count tokens using a declared reference tokenizer for all systems. This enables direct comparison but may not reflect actual system token counts.

The test report MUST declare which option is used. Option B with cl100k_base (GPT-4 tokenizer) as reference is RECOMMENDED for cross-system comparison.

4.4.3. Special Token Handling

The test report MUST specify handling of:

* BOS/EOS tokens (included or excluded from counts)

* System prompt tokens (counted separately or included)

* Tool/function call formatting tokens

4.5. Warm-up Procedures

LLM serving systems require warm-up before reaching steady-state performance. Warm-up effects include JIT compilation, memory allocator initialization, prefix cache population, and batch size ramp-up.

4.5.1. Warm-up Requirements

Before measurement begins, testers MUST:

1. Load the model fully into accelerator memory

2. Process at least 100 requests or 10,000 output tokens, whichever is greater

3. Wait for request queue to drain completely

4. If prefix caching is enabled and being tested, populate the cache with representative prefixes

4.5.2. Warm-up Verification

Testers SHOULD verify warm-up completion by:

1. Measuring latency for a probe request before and after warm-up

2. Confirming latency stabilization (less than 10% variation across consecutive probe requests)

4.5.3. Cold Start Measurement

When cold start performance is being measured (Model Load Time, Cold Start Latency), warm-up MUST be skipped. The test report MUST clearly indicate cold start measurement.

4.6. Streaming Protocol

LLM serving systems deliver tokens via streaming protocols. The choice of protocol affects timing measurement.

4.6.1. Supported Protocols

This methodology supports:

Server-Sent Events (SSE): HTTP-based streaming. Each event contains one or more tokens. RECOMMENDED for Application Gateway testing.

WebSocket: Bidirectional streaming. Each message contains one or more tokens.

gRPC streaming: Binary streaming protocol. Each message contains one or more tokens. RECOMMENDED for Model Engine testing.

4.6.2. Token Chunking

Streaming protocols may deliver multiple tokens per chunk due to batching or network buffering. The test report MUST specify:

* Protocol used

* Whether each chunk contains exactly one token or potentially multiple tokens

* How multi-token chunks are handled for ITL calculation

4.6.3. ITL Calculation with Chunked Delivery

When chunks contain multiple tokens:

Option A - Chunk timing: Measure inter-chunk latency. Report as "Time Between Chunks" rather than ITL. Note chunk size distribution.

Option B - Distributed timing: Distribute chunk arrival time across tokens. If a chunk with N tokens arrives at time T, assign arrival time T to all N tokens. This understates ITL variance.

Option C - Server-side timing: Use server-reported per-token timestamps if available. This measures ITL independent of network effects.

The test report MUST declare which option is used. Option C is RECOMMENDED when available.

4.7. Clock Synchronization

Accurate timing requires synchronized clocks between load generator and SUT, and between distributed SUT components.

4.7.1. Single-Machine Testing

When load generator and SUT run on the same machine, clock synchronization is inherent. This configuration is RECOMMENDED for Model Engine testing.

4.7.2. Distributed Testing

When load generator and SUT are on different machines:

* NTP synchronization MUST achieve accuracy of 10ms or better
* PTP synchronization SHOULD be used when sub-millisecond accuracy is required
* The test report MUST state the synchronization method and estimated accuracy

4.7.3. Network Latency Measurement

For Application Gateway testing where network latency is significant:

* Testers SHOULD measure and report network RTT separately
* Testers MAY subtract estimated network latency from TTFT to isolate server-side processing time
* Any latency adjustment MUST be documented in the test report

4.7.4. Timestamp Format

All timestamps MUST be recorded in a format with at least millisecond precision. ISO 8601 with milliseconds (YYYY-MM-DDTHH:MM:SS.sssZ) or Unix epoch with milliseconds is RECOMMENDED.

4.8. Safety and Guardrail Configuration

Production LLM deployments include safety systems that affect performance. Benchmarking MUST account for these systems.

4.8.1. Guardrail Disclosure

The test report MUST disclose:

* Whether input content filtering is enabled
* Whether output content filtering is enabled
* Names of safety systems if known (e.g., "Llama Guard")
* Whether any requests were refused during testing

4.8.2. Production-Representative Testing

For Application Gateway benchmarking intended to represent production performance:

* Safety systems SHOULD be enabled in their default configuration
* The test report MUST note if safety systems are disabled
* Testers SHOULD run comparative tests with safety enabled and disabled to quantify overhead

5. Benchmarking Tests

This section defines benchmarking tests. Each test includes: objective, setup parameters, procedure, measurements, and reporting format.

5.1. Time to First Token

5.1.1. Objective

To determine the latency from request submission to first token receipt under varying load conditions. TTFT measures perceived responsiveness for interactive applications.

5.1.2. Setup Parameters

The following parameters MUST be defined:

5.1.2.1. Workload Parameters

Workload: One of the standard workloads (Section 4.3.2) or a fully specified custom workload.

Request count: Total number of requests to execute. MUST be at

least 1000 for P99 measurement, 10000 for P99.9.

5.1.2.2. Load Parameters

Load model: Open-loop or closed-loop.

For open-loop:

Arrival rate: Requests per second.

Arrival distribution: Poisson (REQUIRED), uniform, or bursty.

For closed-loop:

Concurrency: Number of concurrent requests maintained.

5.1.2.3. System Parameters

SUT configuration: Model Engine, Application Gateway, or Compound System.

Model identifier: Model name, version, and quantization if applicable.

Hardware: Accelerator type, count, and memory.

Prefix caching: Enabled or disabled.

5.1.3. Procedure

1. Configure the SUT with specified parameters.
2. Complete warm-up procedure (Section 4.5).
3. Begin load generation at the specified arrival rate or concurrency.
4. For each request:
    a. Record submission timestamp (T_submit)
    b. Record first token arrival timestamp (T_first)
    c. Calculate TTFT = T_first - T_submit
    d. Record input token count
5. Continue until request count is reached.

6. Compute distribution statistics.

5.1.3.1. First Token Definition

The first token is defined as the first content token received, excluding:

* Empty tokens or whitespace-only tokens
* Protocol overhead (SSE event markers, JSON framing)
* Metadata tokens (token IDs, logprobs if requested separately)

If the system emits non-content tokens before content, the test report MUST note this and specify whether TTFT measures time to any token or time to first content token.

5.1.4. Measurements

5.1.4.1. Primary Measurements

TTFT Percentiles: P50, P90, P95, P99, and P99.9 of TTFT distribution. All percentiles MUST be reported.

TTFT Mean: Arithmetic mean of TTFT values.

TTFT Minimum: Smallest TTFT observed.

TTFT Maximum: Largest TTFT observed.

5.1.4.2. Conditional Measurements

TTFT by input length: When workload has variable input length, report TTFT percentiles bucketed by input length ranges. RECOMMENDED buckets: [0-256), [256-512), [512-1024), [1024-2048), [2048-4096), [4096+) tokens.

Queue wait time: If measurable (server instrumentation), report the queue wait component of TTFT separately.

Prefill latency: If measurable, report the prefill computation component of TTFT separately.

5.1.4.3. Statistical Requirements

For P99 accuracy within 10% relative error at 95% confidence, at least 1000 samples are required. For P99.9, at least 10000 samples. The test report MUST state the sample count.

5.1.5. Reporting Format

The test report MUST include:

5.1.5.1. Configuration Summary

* SUT configuration and boundary
* Model identifier and hardware
* Workload name or full specification
* Load model and parameters
* Request count and test duration
* Warm-up procedure followed
* Prefix caching state
* Guardrail configuration

5.1.5.2. Results Table

The results SHOULD be reported in tabular format:

Metric	Value
Requests	10000
TTFT P50	127 ms
TTFT P90	245 ms
TTFT P95	312 ms
TTFT P99	524 ms
TTFT P99.9	891 ms
TTFT Mean	156 ms
TTFT Min	89 ms
TTFT Max	1243 ms

Table 1: TTFT Results Example

5.1.5.3. TTFT by Input Length

If applicable:

Input Tokens	P50 (ms)	P95 (ms)	P99 (ms)
0-256	95	198	312
256-512	142	287	445
512-1024	198	412	623
1024-2048	312	587	891
2048+	523	912	1243

Table 2: TTFT by Input Length Example

5.1.5.4. Distribution Visualization

Testers SHOULD include a histogram or CDF plot of the TTFT distribution.

5.2. Output Token Throughput

5.2.1. Objective

To determine the maximum rate at which the SUT can generate output tokens while maintaining acceptable latency. This test measures system capacity under load.

5.2.2. Setup Parameters

The following parameters MUST be defined:

5.2.2.1. Workload Parameters

Workload: One of the standard workloads or fully specified custom workload.

Test duration: Minimum 60 seconds. RECOMMENDED 300 seconds for stable measurement.

5.2.2.2. Load Parameters

Load model: Open-loop or closed-loop.

For open-loop:

Arrival rate range: Minimum and maximum request rates to test.

Rate increment: Step size for iterative search.

For closed-loop:

Concurrency range: Minimum and maximum concurrent requests.

Concurrency increment: Step size for iterative search.

5.2.2.3. Latency Constraint (Optional)

TTFT SLO: Maximum acceptable P99 TTFT.

TPOT SLO: Maximum acceptable P99 TPOT.

When specified, throughput is measured as the maximum rate achieving these SLOs.

5.2.3. Procedure

This test employs an iterative search to find maximum throughput.

1. Configure the SUT with specified parameters.
2. Complete warm-up procedure.
3. For each load level (arrival rate or concurrency):
    a. Run load for the specified test duration.
    b. Record all request timings.
    c. Compute throughput as total output tokens divided by test duration.
    d. Compute TTFT and TPOT percentiles.
    e. If latency constraint specified, check SLO compliance.
4. Use binary search to find maximum throughput:
    a. If no latency constraint: find load level where queue grows unboundedly (system saturation).
    b. If latency constraint: find highest load level meeting SLO.
5. Report throughput at the maximum sustainable load level.

5.2.3.1. Saturation Detection

System saturation is detected when:

* Queue depth grows continuously during test duration, OR
* Request completion rate is less than 90% of arrival rate, OR
* P99 latency exceeds 10x the P50 latency at lower load

5.2.3.2. Steady State Verification

At each load level, verify steady state by:

* Confirming queue depth is stable (not growing)

* Confirming throughput is stable across test duration

* Excluding initial ramp-up period (first 10% of duration)

5.2.4. Measurements

5.2.4.1. Primary Measurements

Maximum output token throughput: Output tokens per second at maximum sustainable load. Report with or without latency constraint as specified.

Request throughput: Requests completed per second at maximum load.

Input token throughput: Input tokens processed per second (measures prefill capacity).

5.2.4.2. Efficiency Measurements

Tokens per GPU-second: Output tokens per second divided by GPU count. Enables comparison across different hardware configurations.

Batch utilization: If measurable, report average batch size divided by maximum batch size.

5.2.4.3. Latency at Maximum Throughput

At the maximum sustainable load level, report:

* TTFT P50, P95, P99

* TPOT P50, P95, P99

* End-to-end latency P50, P95, P99

5.2.5. Reporting Format

5.2.5.1. Summary Results

Metric	Value
Max Output Throughput	2847 tok/s
Max Request Throughput	18.2 req/s
Max Input Throughput	5123 tok/s

Sustainable Load	20 req/s
Tokens per GPU-second	356 tok/s/GPU

Table 3: Throughput Summary Example

5.2.5.2. Latency at Maximum Throughput

Metric	P50	P95	P99
TTFT	312 ms	687 ms	1124 ms
TPOT	42 ms	78 ms	134 ms
End-to-End	6.2 s	11.4 s	18.7 s

Table 4: Latency at Maximum Throughput Example

5.3. Throughput-Latency Tradeoff

5.3.1. Objective

To characterize the relationship between throughput and latency across the operating range of the SUT. This test produces a throughput-latency curve revealing system behavior better than point measurements.

5.3.2. Setup Parameters

Workload: One of the standard workloads or fully specified custom workload.

Test duration per point: Minimum 60 seconds per load level.

Load levels: At least 10 load levels spanning from low load (10% of estimated capacity) to saturation.

Load model: Open-loop is REQUIRED for this test. Closed-loop cannot reveal behavior beyond capacity.

5.3.3. Procedure

1. Estimate system capacity using a preliminary throughput test or published specifications.

2. Define load levels: 10%, 20%, 30%, ..., 100%, 110%, 120% of estimated capacity.

3. For each load level in ascending order:

    a. Run load for specified duration.

    b. Record all request timings.

    c. Compute achieved throughput (may differ from offered load at saturation).

    d. Compute latency percentiles.

4. Plot throughput vs latency curves.

5.3.4. Measurements

For each load level, record:

* Offered load (request rate)

* Achieved throughput (output tokens per second)

* TTFT: P50, P95, P99

* TPOT: P50, P95, P99

* End-to-end latency: P50, P95, P99

* Request success rate

* Queue growth indicator (stable/growing)

Derived metrics:

Optimal operating point: Load level achieving highest throughput while meeting specified SLO.

Knee point: Load level where P99 latency exceeds 2x the minimum P99 latency observed.

Saturation point: Load level where achieved throughput first decreases from previous level.

5.3.5. Reporting Format

Offered (r/s)	Achieved (tok/s)	TTFT P50	TTFT P99	TPOT P50	TPOT P99	Success
2	284	95	142	32	41	100%
6	852	102	178	34	48	100%
10	1420	128	267	38	62	100%
14	1988	198	512	48	98	100%
18	2534	378	1234	72	198	99.8%
22	2712	823	3456	142	523	94.1%

Table 5: Throughput-Latency Table Example

Knee point: 14 req/s (TTFT P99 exceeds 2x minimum)

Saturation point: 22 req/s (throughput peaks)

5.4. Inter-Token Latency Distribution

5.4.1. Objective

To characterize the variability of token delivery during the decode phase. ITL distribution determines streaming smoothness experienced by users.

5.4.2. Setup Parameters

Workload: Synthetic-Uniform or Conversation workload RECOMMENDED.

Minimum output length: Requests MUST generate at least 50 output tokens to provide meaningful ITL samples.

Request count: At least 100 requests for per-request statistics, yielding 5000+ ITL samples.

Load level: Specify as percentage of maximum throughput. Multiple load levels RECOMMENDED: 25%, 50%, 75%, 90% of saturation.

Measurement method: Specify per Section 4.6.3 (chunk timing, distributed timing, or server-side timing).

5.4.3. Procedure

1. Configure SUT and complete warm-up.

2. For each load level:

    a. Generate requests at specified load.

    b. For each request, record arrival time of each token after the first.

    c. Calculate ITL_i = T(token_i) - T(token_{i-1}) for each consecutive token pair.

    d. Aggregate ITL samples across all requests.

    e. Calculate per-request jitter (standard deviation of ITL within each request).

    f. Record maximum pause duration per request.

The interval between request submission and first token (TTFT) MUST NOT be included in ITL calculation.

5.4.4. Measurements

5.4.4.1. Aggregate ITL Statistics

ITL Percentiles: P50, P90, P95, P99, P99.9 across all ITL samples.

ITL Mean: Arithmetic mean of all ITL samples.

ITL Standard Deviation: Standard deviation across all samples.

5.4.4.2. Per-Request Statistics

Jitter Distribution: P50, P95, P99 of per-request standard deviation.

Maximum Pause Distribution: P50, P95, P99 of per-request maximum ITL.

5.4.4.3. Distribution Shape

Modality: Whether ITL distribution is unimodal or multimodal. Multimodal distributions indicate distinct operating regimes (e.g., batching effects).

Tail behavior: Characterize tail (exponential, heavy-tailed). Report the ratio P99/P50 as a tail heaviness indicator.

5.4.5. Reporting Format

Metric	Value
ITL Samples	15234
ITL P50	38 ms
ITL P90	52 ms
ITL P95	67 ms
ITL P99	124 ms
ITL P99.9	312 ms
ITL Mean	42 ms
ITL Std Dev	28 ms
P99/P50 Ratio	3.26

Table 6: ITL Results Example

5.5. Concurrent Request Capacity

5.5.1. Objective

To determine the maximum number of concurrent requests the SUT can maintain while meeting latency objectives. This test measures memory capacity and scheduling limits.

5.5.2. Setup Parameters

Workload: Synthetic-Uniform RECOMMENDED for controlled testing.

Fixed output length: Use fixed output length (e.g., 256 tokens) to ensure all requests have similar duration.

Initial concurrency: Starting number of concurrent requests (e.g., 8).

Maximum concurrency: Upper bound for search (e.g., 512).

Success criteria: Request completion rate >= 99%, TTFT P99 <= specified threshold, and no out-of-memory errors.

5.5.3. Procedure

This test employs binary search to find maximum concurrent capacity.

1. Configure SUT and complete warm-up.
2. Set concurrency = initial concurrency.
3. For each concurrency level:
    a. Submit [concurrency] requests simultaneously.
    b. Maintain concurrency: when a request completes, immediately submit a replacement.
    c. Run for at least 60 seconds or 100 request completions per slot, whichever is longer.
    d. Record completion rate, latency percentiles, and any errors.
    e. Check success criteria.
4. Binary search:
    a. If success criteria met: increase concurrency toward maximum.
    b. If success criteria not met: decrease concurrency.
    c. Continue until convergence.
5. Report maximum concurrency meeting success criteria.

5.5.4. Measurements

Maximum concurrent requests: Highest concurrency meeting success criteria.

Achieved throughput at maximum: Output tokens per second at maximum concurrency.

Tokens in flight at maximum: Approximate total tokens (input + output so far) across all concurrent requests.

5.5.5. Reporting Format

Concurrency	Completion	TTFT P99	TPOT P99	Errors	Status
8	100%	142 ms	38 ms	0	Pass
16	100%	178 ms	42 ms	0	Pass
32	100%	267 ms	52 ms	0	Pass
64	99.7%	523 ms	78 ms	0	Pass
128	97.2%	1234 ms	156 ms	3	Fail

Table 7: Capacity Search Results Example

Maximum concurrent requests meeting criteria: 64

5.6. Scheduling Fairness

5.6.1. Objective

To evaluate how equitably the SUT allocates resources across concurrent requests with different characteristics. This test reveals head-of-line blocking, starvation, and priority effects.

5.6.2. Setup Parameters

Workload: Synthetic-Skewed REQUIRED. The high length variance creates fairness-sensitive conditions.

Request classes: Define two or more request classes:

* Short requests: Input [64, 256] tokens, output [32, 128] tokens
* Long requests: Input [1024, 4096] tokens, output [256, 1024] tokens

Class mix: Ratio of request classes (e.g., 80% short, 20% long).

Load level: 70-90% of saturation throughput RECOMMENDED to create contention.

Request count: At least 500 requests per class.

5.6.3. Procedure

1. Configure SUT and complete warm-up.
2. Measure baseline: performance of each class in isolation at same total load.
3. Generate mixed workload with specified class ratio.
4. Run at specified load level for at least 300 seconds.
5. For each request, record class membership, submission time, first token time, completion time.
6. Compute per-class statistics and fairness metrics.

5.6.4. Measurements

Per-class latency: TTFT P50, P95, P99 for each request class.

Latency inflation: (Mixed workload TTFT) / (Isolated TTFT) per class.

Jain's Fairness Index: $J = (\sum(x_i))^2 / (n * \sum(x_i^2))$ where x_i is normalized latency. $J = 1.0$ indicates perfect fairness. $J < 0.9$ indicates significant unfairness.

Starvation rate: Fraction of requests waiting longer than 5x the median wait time for their class.

5.6.5. Reporting Format

Class	Count	TTFT P50	TTFT P99	TPOT P50	TPOT P99
Short	4012	89 ms	234 ms	35 ms	67 ms
Long	988	312 ms	1234 ms	42 ms	89 ms

Table 8: Per-Class Results Example

Metric	Value
Jain's Fairness Index	0.87
Short Class Starvation	0.3%
Long Class Starvation	2.1%

Table 9: Fairness Metrics Example

5.7. Prefix Cache Effectiveness

5.7.1. Objective

To evaluate the performance benefit of prefix caching under workloads with shared prefixes. This test quantifies TTFT reduction from cache hits.

5.7.2. Setup Parameters

Workload: Code Completion workload RECOMMENDED (high prefix sharing).

Shared prefix: Define a prefix shared across requests.

Prefix length: Length in tokens of shared prefix.

Sharing fraction: Percentage of requests sharing the prefix.

Comparison mode: Test MUST run in two configurations: cache disabled (baseline) and cache enabled.

5.7.3. Procedure

1. Configure SUT with cache disabled.
2. Complete warm-up (without populating prefix cache).
3. Run workload, record TTFT for all requests.
4. Enable prefix cache.
5. Optionally pre-populate cache with shared prefix.
6. Run identical workload, record TTFT for all requests.

7. Compare results.

5.7.4. Measurements

TTFT without cache: P50, P95, P99 with caching disabled.

TTFT with cache: P50, P95, P99 with caching enabled.

TTFT reduction: (TTFT_no_cache - TTFT_cache) / TTFT_no_cache as percentage.

Cache hit rate: Fraction of prefix tokens served from cache.

Throughput improvement: Percentage increase from caching.

5.7.5. Reporting Format

Configuration	TTFT P50	TTFT P95	TTFT P99
Cache Disabled	312 ms	423 ms	534 ms
Cache (Cold)	134 ms	198 ms	267 ms
Cache (Warm)	98 ms	156 ms	212 ms

Table 10: Cache Effectiveness Example

5.8. Memory Pressure Behavior

5.8.1. Objective

To characterize SUT behavior when memory resources are constrained, including preemption, swapping, and degradation patterns.

5.8.2. Setup Parameters

Workload: Long Context workload RECOMMENDED to create memory pressure.

Oversubscription level: Percentage above maximum capacity (e.g., 110%, 125%, 150%).

5.8.3. Procedure

1. Determine maximum concurrent capacity from Section 5.5.

2. Configure SUT and complete warm-up.

3. For each oversubscription level:

   a. Submit requests at concurrency exceeding capacity.

   b. Run for at least 120 seconds.

   c. Monitor request completions, preemption events, latency.

   d. Record any OOM errors or system failures.

4. Analyze degradation patterns.

5.8.4. Measurements

Completion rate: Percentage of requests completing successfully at each level.

Preemption rate: Fraction of requests preempted at least once.

Preemption recovery rate: Fraction of preempted requests that eventually complete.

Preemption loss: Average tokens discarded per preemption event.

5.8.5. Reporting Format

Oversub Level	Complete	Preempt	Fail Rate	TTFT P99
100% (base)	99.7%	0%	0.3%	523 ms
110%	98.2%	5.2%	1.8%	789 ms
125%	94.5%	18.7%	5.5%	1456 ms
150%	82.3%	42.1%	17.7%	3234 ms

Table 11: Memory Pressure Degradation Example

5.9. Long Context Scaling

5.9.1. Objective

To characterize how latency and throughput scale with context length.

5.9.2. Setup Parameters

Workload: Long Context workload REQUIRED.

Context length range: Sequence of lengths to test (e.g., 1K, 2K, 4K, 8K, 16K, 32K, 64K, 128K tokens).

Fixed output length: Use consistent short output (256 tokens) to isolate prefill impact.

Load model: Closed-loop with low concurrency (1-4).

Requests per length: At least 20 requests per context length.

5.9.3. Procedure

1. Configure SUT and complete warm-up with short-context requests.
2. For each context length in ascending order:
    a. Generate requests with specified input length.
    b. Submit requests at low concurrency.
    c. Record TTFT and total latency for each request.
3. Analyze scaling behavior and fit to scaling models.

5.9.4. Measurements

Per-length latency: TTFT Mean, P50, P95 for each context length.

Prefill scaling: Time per input token (TTFT / input_length).

Scaling exponent: Fit exponent k where TTFT proportional to context_length^k.

Throughput at length: Maximum throughput achievable at each context length.

5.9.5. Reporting Format

Context (tokens)	TTFT Mean	TTFT P95	ms/1K tokens
1024	89 ms	112 ms	76
4096	289 ms	367 ms	63

16384	1023 ms	1287 ms	59
65536	4234 ms	5123 ms	62
131072	9123 ms	11234 ms	68

Table 12: Long Context Scaling Example

Best fit: Linear (R^2 = 0.9987), ~68 microseconds per input token

5.10. Guardrail Overhead

5.10.1. Objective

To quantify the latency impact of safety systems and content filtering.

5.10.2. Setup Parameters

Workload: Conversation workload RECOMMENDED.

Content mix: Use benign content to measure processing overhead.

Configurations to compare: The following configurations should be tested:

* Baseline: All guardrails disabled (if possible)
* Input filtering only
* Output filtering only
* Full filtering: All production guardrails enabled

Load levels: Test at 25%, 50%, 75% of capacity.

5.10.3. Procedure

1. Configure SUT with baseline (no guardrails).
2. Complete warm-up and run workload at each load level.
3. Enable each guardrail configuration and repeat.
4. Compare results across configurations.

5.10.4. Measurements

Per-configuration latency: TTFT P50, P95, P99 and End-to-end latency for each configuration.

Input filter overhead: TTFT(input_filter) - TTFT(baseline)

Total guardrail overhead: End-to-end(full) - End-to-end(baseline)

Throughput reduction: Percentage reduction from guardrails.

5.10.5. Reporting Format

Configuration	TTFT P50	TTFT P99	E2E P50	E2E P99
Baseline	98 ms	234 ms	4.2 s	8.7 s
Input Filter	112 ms	267 ms	4.3 s	8.9 s
Output Filter	101 ms	242 ms	4.8 s	9.8 s
Full Filter	118 ms	289 ms	5.0 s	10.2 s

Table 13: Guardrail Overhead Example

Configuration	Max Throughput	Reduction
Baseline	2867 tok/s	-
Input Filter	2756 tok/s	-3.9%
Output Filter	2412 tok/s	-15.9%
Full Filter	2289 tok/s	-20.2%

Table 14: Throughput Impact Example

6. Multi-System Comparison Guidelines

When comparing multiple SUTs:

6.1. Equivalence Requirements

Testers MUST ensure:

* Identical workload (same requests in same order with same seeds)
* Equivalent SUT boundary (all systems at same boundary)
* Comparable hardware (or normalize by hardware capability)
* Same load model and parameters

6.2. Normalization

When hardware differs:

* Report tokens per GPU-second (normalized by GPU count)
* Report cost-normalized throughput (tokens per dollar-hour)
* Clearly state normalization method

6.3. Statistical Significance

For comparative claims:

* Report confidence intervals for key metrics
* Conduct multiple independent runs (at least 3)
* Use appropriate statistical tests for comparison

6.4. Fair Comparison Checklist

Before publishing comparative results, verify:

* Same workload specification
* Same test duration
* Same warm-up procedure
* Same success criteria
* Both systems tested at same time (if using shared resources)
* Both systems in production-representative configuration

* Differences in configuration explicitly noted

7. Security Considerations

Benchmarking methodology intersects with security in several ways.

7.1. Side-Channel Risks

Benchmark results may reveal:

* System capacity limits useful for DoS planning

* Timing patterns enabling cache probing attacks

* Memory pressure thresholds for resource exhaustion

Operators SHOULD consider whether to publish detailed capacity information publicly.

7.2. Benchmark Gaming

Systems may be optimized specifically for benchmark workloads in ways that do not generalize:

* Detecting benchmark patterns and applying special handling

* Caching benchmark-specific prefixes

* Prioritizing benchmark-like requests

Testers SHOULD vary workloads and verify results with production traffic samples.

7.3. Adversarial Workloads

This methodology uses benign workloads. Adversarial inputs (jailbreak attempts, prompt injections) may have different performance characteristics due to guardrail processing.

Testing with adversarial workloads requires additional ethical and safety considerations not covered here.

7.4. Resource Exhaustion

Memory pressure tests (Section 5.8) intentionally push systems beyond capacity. Testers SHOULD:

* Conduct such tests on isolated systems

* Have recovery procedures ready

* Monitor for cascading failures

8. References

8.1. 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>.

[LLM-TERMS] Gaikwad, M., "Benchmarking Terminology for Large Language Model Serving", Work in Progress, Internet-Draft, draft-gaikwad-llm-benchmarking-terminology-00, January 2026, <https://datatracker.ietf.org/doc/draft-gaikwad-llm-benchmarking-terminology/>.

8.2. 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>.

[RFC3511] Hickman, B., Newman, D., Tadjudin, S., and T. Martin, "Benchmarking Methodology for Firewall Performance", RFC 3511, DOI 10.17487/RFC3511, April 2003, <https://www.rfc-editor.org/info/rfc3511>.

[VLLM] Kwon, W., "Efficient Memory Management for Large Language Model Serving with PagedAttention", Proceedings of SOSP 2023, DOI 10.1145/3600006.3613165, 2023, <https://doi.org/10.1145/3600006.3613165>.

[SARATHI] Agrawal, A., "Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve", Proceedings of OSDI 2024, 2024.

Appendix A. Reference Workload Specifications

This appendix provides complete specifications for standard workloads.

A.1. Synthetic-Uniform Workload

Purpose: Controlled baseline with minimal variance

A.1.1. Input Specification

Distribution: Uniform

Minimum: 128 tokens

Maximum: 512 tokens

Mean: 320 tokens

Content: Random token IDs from vocabulary

A.1.2. Output Specification

Distribution: Uniform

Minimum: 64 tokens

Maximum: 256 tokens

Mean: 160 tokens

Control: max_tokens parameter

A.1.3. Other Parameters

System prompt: None

Prefix sharing: None

Temperature: 0.0 (deterministic)

Stop sequences: None

A.1.4. Generation Method

Python pseudocode:

```
   def generate_synthetic_uniform(n_requests, seed=42):
       rng = random.Random(seed)
       requests = []
       for i in range(n_requests):
           input_len = rng.randint(128, 512)
           output_len = rng.randint(64, 256)
           input_tokens = [rng.randint(0, 100255)
                           for _ in range(input_len)]
           requests.append({
               'input_tokens': input_tokens,
               'max_tokens': output_len,
               'temperature': 0.0
           })
       return requests
```

A.2. Synthetic-Skewed Workload

Purpose: Test scheduling with high length variance

A.2.1. Input Specification

Distribution: Log-normal

mu: 5.5 (in log space)

sigma: 1.0 (in log space)

Minimum: 32 tokens (floor)

Maximum: 4096 tokens (cap)

Median: ~245 tokens

Mean: ~405 tokens

A.2.2. Output Specification

Distribution: Log-normal

mu: 4.5 (in log space)

sigma: 1.2 (in log space)

Minimum: 16 tokens (floor)

Maximum: 2048 tokens (cap)

A.3. Conversation Workload

Purpose: Realistic interactive chat patterns

A.3.1. Data Source

Dataset: ShareGPT (vicuna_cleaned subset)

Version: 2023-04-12

Preprocessing: Filter conversations with 1+ assistant turns

A.3.2. Length Statistics (Reference)

Input tokens:

* P50: 156
* P95: 892
* P99: 2134

Output tokens:

* P50: 234
* P95: 789
* P99: 1567

A.4. Code Completion Workload

Purpose: Test prefix caching with code context

A.4.1. Data Source

Dataset: The Stack (Python, JavaScript, TypeScript subset)

Preprocessing: Extract function-level completions

A.4.2. Prefix Sharing Pattern

* 10 unique repository contexts
* Each 512-1024 tokens
* 80% of requests share one of these prefixes

* Distribution: Zipf with s=1.5

A.5. Long Context Workload

Purpose: Test long-context handling

A.5.1. Input Specification

Distribution: Uniform over target lengths

Target lengths: [8192, 16384, 32768, 65536, 131072] tokens

Structure: [document][question]

Document: Fills target length minus 100 tokens

Question: Fixed ~100 token question about document

A.5.2. Output Specification

Distribution: Fixed

Length: 256 tokens

Control: max_tokens = 256

Appendix B. Timing Measurement Reference

This appendix provides detailed guidance for timing measurements.

B.1. TTFT Measurement Points

B.1.1. HTTP/SSE Measurement

Client-side TTFT:

T_submit: time of sending final byte of HTTP request

T_first: time of receiving first data event with content token

T_first is when the complete "data:" line is received and parsed, not when the first byte of the response arrives.

B.1.2. gRPC Streaming Measurement

T_submit: time of sending request message

T_first: time of receiving first response message with token

B.1.3. Server-Side Measurement

If server instrumentation available:

T_submit: time request enters inference queue

T_first: time first token exits model forward pass

Server-side excludes network latency but may include internal queue time.

B.2. ITL Measurement with SSE

SSE delivery may batch multiple tokens per event due to server-side batching, TCP buffering, or client-side buffering.

B.2.1. Recommended Approach

1. First, characterize delivery pattern (tokens per chunk)
2. If single-token chunks dominate (>90%): use direct measurement
3. If multi-token chunks common: prefer server timestamps
4. If server timestamps unavailable: use chunk timing and document

B.3. Clock Synchronization Methods

B.3.1. NTP Synchronization

1. Both machines sync to same NTP server
2. Verify offset: ntpq -p (check offset column)
3. Acceptable offset: < 10ms for most LLM benchmarking
4. Document NTP server and measured offset

B.3.2. PTP Synchronization

For sub-millisecond accuracy:

1. Use PTP-capable network hardware
2. Configure ptp4l on Linux systems
3. Acceptable offset: < 1 microsecond

B.3.3. Single-Machine Alternative

Recommended for Model Engine testing:

1. Run load generator on same machine as SUT
2. Use loopback network interface
3. Clock synchronization inherent
4. Eliminates network latency from measurement

Appendix C. Reporting Templates

C.1. Minimum Viable Report

For quick comparisons, include at minimum:

```
=== LLM Benchmark Report (Minimum) ===

System Identification:
- Model: [model name and version]
- Hardware: [GPU type] x [count]
- Software: [inference engine and version]
- SUT Boundary: [Model Engine | Gateway | Compound]

Test Configuration:
- Workload: [workload name]
- Load Model: [open-loop rate | closed-loop concurrency]
- Request Count: [N]
- Test Duration: [seconds]

Key Results:
- TTFT P50: [value] ms
- TTFT P99: [value] ms
- TPOT P50: [value] ms
- TPOT P99: [value] ms
- Max Throughput: [value] tok/s
- Throughput at P99 TTFT < 500ms: [value] tok/s

Notes:
- [Any deviations from methodology]
- [Guardrail configuration]

=== End Report ===
```

C.2. Full Report Template

A complete benchmark report should include the following sections:

1. System Identification (model, hardware, software)
2. Test Configuration (workload, load, execution parameters)
3. Results (latency summary, throughput summary, success metrics)
4. Detailed Results (per-test tables and visualizations)
5. Methodology Compliance (tests performed, deviations, limitations)
6. Reproduction Information (test harness, configuration, data)

Acknowledgements

This document draws on the structure and approach established by RFC 3511 for firewall benchmarking methodology. The author thanks the Benchmarking Methodology Working Group for their foundational work in network device benchmarking.

Author's Address

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