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

M. Gaikwad
Independent Researcher

January 2026

Performance Benchmarking Profiles for Large Language Model Serving Systems

draft-gaikwad-llm-benchmarking-profiles-00

Abstract

This document defines performance benchmarking profiles for Large Language Model (LLM) serving systems. Profiles bind the terminology defined in draft-gaikwad-llm-benchmarking-terminology and the procedures described in draft-gaikwad-llm-benchmarking-methodology to concrete architectural roles and workload patterns. Each profile clarifies the System Under Test (SUT) boundary, measurement points, and interpretation constraints required for reproducible and comparable benchmarking.

This document specifies profiles only. It does not define new metrics, benchmark workloads, 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

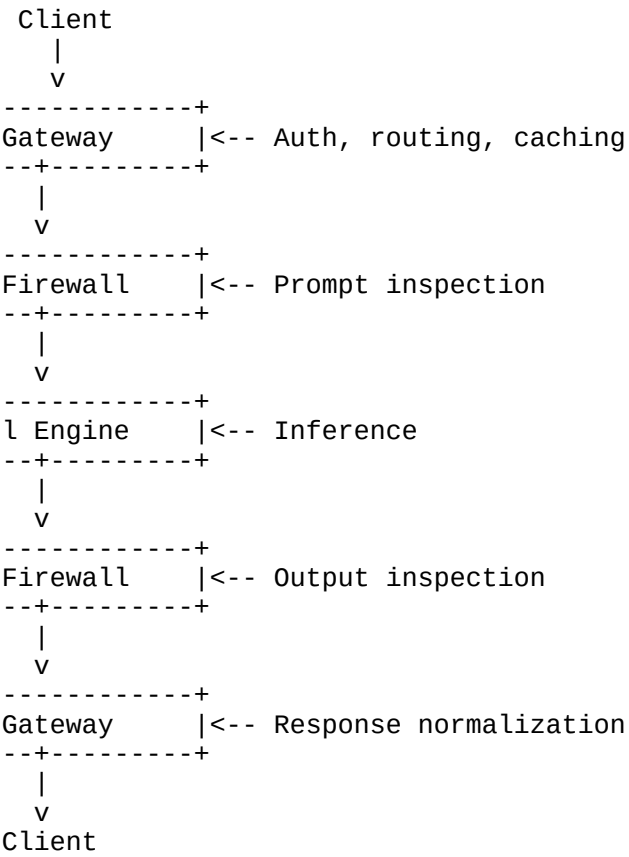
Table of Contents

1. Introduction

LLM serving systems are rarely monolithic. Production deployments typically compose multiple infrastructural intermediaries before a request reaches a Model Engine. A request may pass through an API gateway for authentication, an AI firewall for prompt inspection, a load balancer for routing, and finally arrive at an inference engine. Each component adds latency and affects throughput.

Performance metrics such as Time to First Token (TTFT) or throughput are boundary dependent. A TTFT measurement taken at the client includes network latency, gateway processing, firewall inspection, queue wait time, and prefill computation. The same measurement taken at the engine boundary includes only queue wait and prefill. Without explicit boundary declaration, reported results cannot be compared.

This document addresses this ambiguity by defining benchmarking profiles: standardized descriptions of SUT boundaries and their associated performance interpretation rules. Section 4 defines four infrastructure profiles that specify what component is being measured. Section 5 defines workload profiles that specify how that component is tested. Section 6 then shows how to attribute latency across composed systems using delta measurement.



Each layer adds latency. Benchmarks must declare which layers are included.

Figure 1: Typical LLM Serving Stack

1.1. Terminology Alignment

This document uses metrics defined in [I-D.gaikwad-llm-benchmarking-terminology]. The following table maps profile-specific terms to their normative definitions.

Term Used in Profiles	Terminology Draft Reference
TTFT	Time to First Token
ITL	Inter-Token Latency
TPOT	Time per Output Token
Queue Residence Time	Queue Wait Time
FRR	False Refusal Rate
Guardrail Overhead	Guardrail Processing Overhead
Task Completion Latency	Task Completion Latency
Goodput	Goodput

Table 1: Terminology Mapping

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.

3. Profile Taxonomy

Profiles divide into two categories that serve orthogonal purposes. Conflating them produces misleading benchmarks.

3.1. Infrastructure Profiles

Infrastructure Profiles define what is being tested. They specify the SUT boundary: where measurements start and end, what components are included, and what is excluded.

Profile	SUT Boundary	Primary Question Answered
Model Engine	Inference runtime only	How fast can this engine generate tokens?
AI Gateway	API intermediary layer	What overhead does the gateway add?
AI Firewall	Security inspection layer	What latency and accuracy does inspection cost?
Compound System	End-to-end orchestration	How long does it take to complete a task?

Table 2: Infrastructure Profiles

The choice of infrastructure profile determines which metrics are meaningful. Measuring "AI Firewall throughput" in tokens per second conflates firewall performance with downstream engine performance. The firewall does not generate tokens; it inspects them. Appropriate firewall metrics include inspection latency, detection rate, and false positive rate.

3.2. Workload Profiles

Workload Profiles define how the SUT is tested. They specify traffic patterns, request characteristics, and arrival models. Workload profiles are independent of infrastructure profiles.

Profile	Traffic Pattern	Applicable To
Chatbot Workload	Multi-turn, streaming, human-paced	Engine, Gateway, Firewall, Compound
Compound Workflow	Multi-step, tool-using, machine-paced	Compound System primarily

Table 3: Workload Profiles

A Chatbot Workload can be applied to a Model Engine (measuring raw inference speed), an AI Gateway (measuring gateway overhead under conversational traffic), or a Compound System (measuring end-to-end chat latency including retrieval). The infrastructure profile determines the measurement boundary; the workload profile determines the traffic shape.

Conflating infrastructure and workload profiles produces non-comparable results. "Chatbot benchmark on Gateway A" versus "Chatbot benchmark on Engine B" compares different things. The former includes gateway overhead; the latter does not. Valid comparison requires either:

* Same infrastructure profile, different implementations (Gateway A vs Gateway B)

* Same implementation, different workload profiles (Chatbot vs Compound Workflow on Engine A)

Cross-profile comparisons require explicit delta decomposition (Section 6).

3.3. Profile Selection Guidance

If you want to measure...	Use Infrastructure Profile	Apply Workload Profile
Raw model inference speed	Model Engine	Chatbot or synthetic
Gateway routing overhead	AI Gateway	Match production traffic
Security inspection cost	AI Firewall	Mixed benign/ adversarial
End-to-end agent latency	Compound System	Compound Workflow
Full-stack production performance	Composite (see Section 7)	Match production traffic

Table 4: Profile Selection Guide

4. Infrastructure Profiles

4.1. Model Engine Profile

4.1.1. Definition and Concepts

A Model Engine is the runtime responsible for executing LLM inference. Before specifying the benchmark boundary, understanding three core operations is necessary:

Prefill (also called prompt processing): The engine processes all input tokens in parallel to build initial hidden states. Prefill is compute-bound and benefits from parallelism. Prefill latency scales with input length but can be reduced by adding more compute.

Decode (also called autoregressive generation): The engine generates output tokens one at a time, each depending on all previous tokens. Decode is memory-bandwidth-bound because each token requires reading the full model weights. Decode latency per token is relatively constant regardless of batch size, but throughput increases with batching.

KV Cache: To avoid recomputing attention over previous tokens, the engine stores key-value pairs from prior tokens. The KV cache grows with sequence length and consumes GPU memory. Cache management (allocation, eviction, swapping to CPU) directly affects how many concurrent sequences the engine can handle.

These three operations determine the fundamental performance characteristics:

* TTFT depends primarily on prefill time plus any queue wait
* ITL depends on decode time per token
* Maximum concurrency depends on KV cache capacity
* Throughput depends on batching efficiency during decode

4.1.2. Boundary Specification

Included in SUT:

* Model weights and inference kernels
* Prefill and decode computation
* Batch formation and scheduling logic
* KV cache allocation, eviction, and swapping

* Speculative decoding (if enabled)

* Quantization and precision handling

Excluded from SUT:

* Network transport beyond local interface

* Authentication and authorization

* Policy enforcement and content inspection

* Request routing between multiple engines

* Protocol translation (handled by gateway)

4.1.3. Architecture Variants

Model Engines exist in several architectural configurations that affect measurement interpretation.

4.1.3.1. Monolithic Architecture

Prefill and decode execute on the same hardware. This is the simplest configuration and the most common in single-GPU deployments.

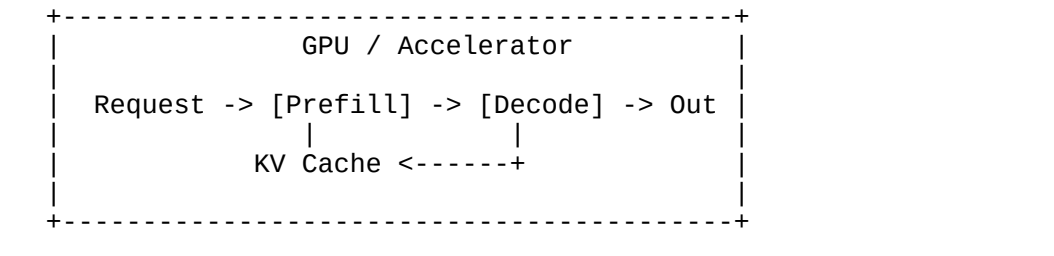


Figure 2: Monolithic Engine Architecture

Timestamp mapping:

Symbol	Event
t6	Request enters engine queue
t6a	Prefill computation begins (batch slot acquired)
t7	First output token generated
t8	Last output token generated

Table 5

Derived metrics:

Queue residence = t6a - t6
Prefill latency = t7 - t6a
Engine TTFT = t7 - t6
Generation time = t8 - t7

4.1.3.2. Disaggregated Architecture

Prefill and decode execute on separate hardware pools. Prefill nodes are optimized for compute throughput; decode nodes are optimized for memory bandwidth. After prefill completes, the KV cache must transfer across the network to the decode pool.

This architecture appears in published systems including DistServe [DISTSERVE] and Mooncake [MOONCAKE], and in open-source projects such as llm-d.

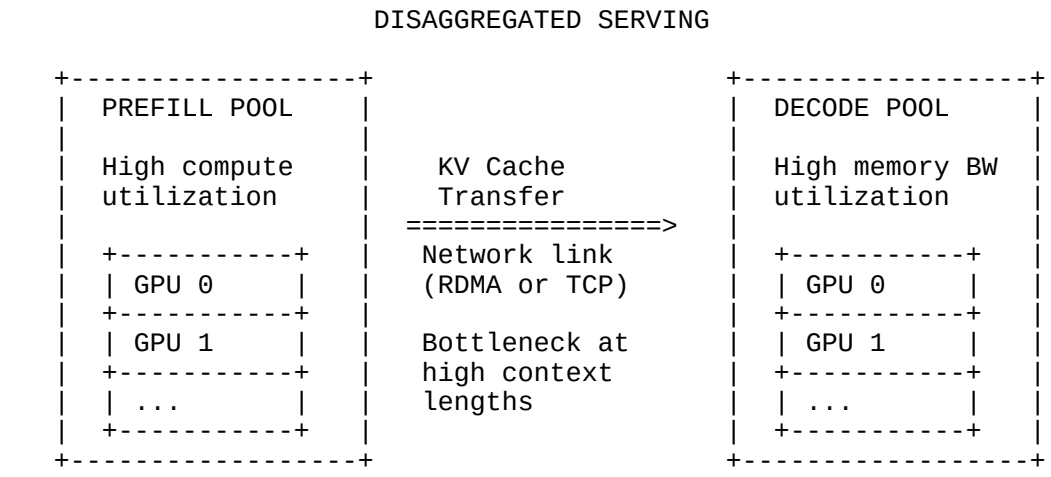


Figure 3: Disaggregated Serving Architecture

The KV transfer phase (t7a) does not exist in monolithic deployments. This phase can become the bottleneck for long contexts.

KV Transfer Constraint:

Transfer time depends on context length and network bandwidth:

```
KV_transfer_time = (context_length * kv_bytes_per_token) / effective_bandwidth
```

Where:

* context_length = input tokens processed

* kv_bytes_per_token = 2 * num_layers * head_dim * num_heads * bytes_per_element

* effective_bandwidth = min(network_bandwidth, memory_bandwidth) * efficiency

Bandwidth Saturation Threshold: The context length at which KV transfer time exceeds prefill compute time. Beyond this threshold, adding more prefill compute does not reduce TTFT.

```
Configuration:
  Model: 70B parameters
  KV cache: 80 layers, 128 heads, 128 dim, BF16
  KV bytes per token: 2 * 80 * 128 * 128 * 2 = 5.24 MB
  Inter-pool bandwidth: 400 Gbps = 50 GB/s effective

At 4K context:
  KV transfer = 4096 * 5.24 MB / 50 GB/s = 430 ms

At 32K context:
  KV transfer = 32768 * 5.24 MB / 50 GB/s = 3.44 s

If prefill compute takes 500ms for 32K tokens:
  Bottleneck shifts to KV transfer at ~4.8K tokens
```

Figure 4: KV Transfer Example Calculation

Testers benchmarking disaggregated architectures MUST report:

Parameter	Description
Pool configuration	Number and type of prefill vs decode accelerators
KV transfer mechanism	RDMA, TCP, or other; theoretical bandwidth
KV bytes per token	Calculated from model architecture
Observed transfer latency	Measured, not calculated
Bandwidth saturation threshold	Context length where transfer becomes bottleneck
TTFT boundary	Whether reported TTFT includes KV transfer

Table 6

Results from disaggregated and monolithic deployments MUST NOT be directly compared without explicit architectural notation.

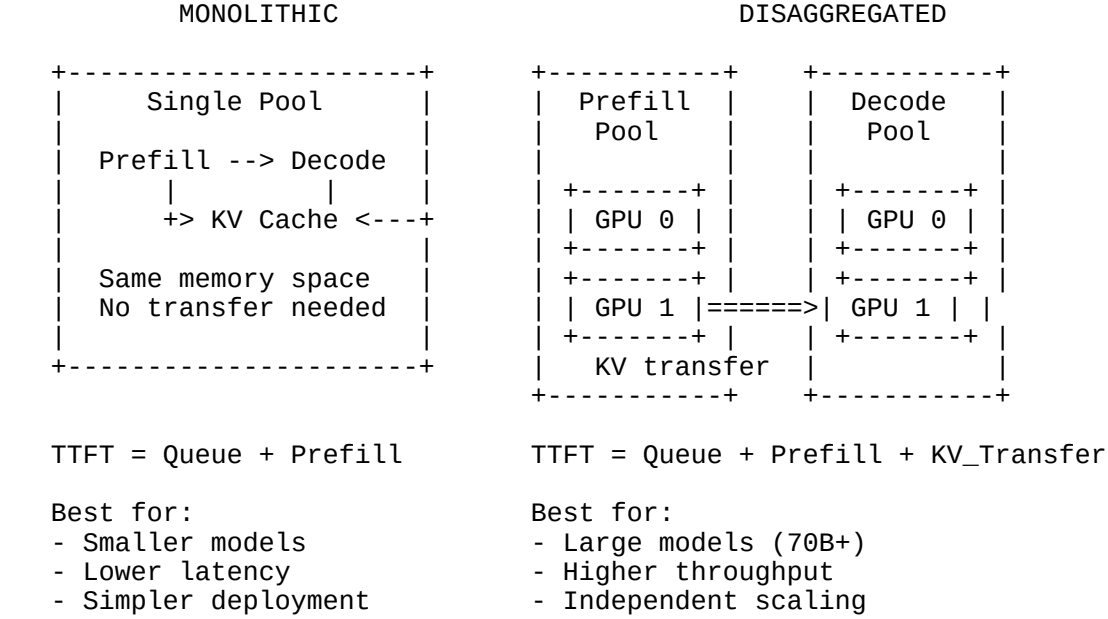


Figure 5: Monolithic vs Disaggregated Comparison

4.1.3.3. Distributed Architecture

Model sharded across multiple accelerators using tensor parallelism (TP), pipeline parallelism (PP), or expert parallelism (EP for mixture-of-experts models).

Testers MUST report:

* Parallelism strategy and degree (e.g., TP=8, PP=2)

* Interconnect type (NVLink, PCIe, InfiniBand)

* Collective communication overhead if measurable

4.1.4. Configuration Disclosure

Testers MUST disclose:

Configuration	Example Values	Why It Matters
Model precision	FP16, BF16, INT8, FP8	Affects throughput, memory, and quality
Quantization method	GPTQ, AWQ, SmoothQuant	Different speed/ quality tradeoffs
Batch strategy	Static, continuous, chunked prefill	Affects latency distribution
Max batch size	64 requests	Limits concurrency
Max sequence length	8192 tokens	Limits context window
KV cache memory	24 GB	Limits concurrent sequences

Table 7

4.1.4.1. Speculative Decoding

Speculative decoding uses a smaller draft model to propose multiple tokens, then verifies them in parallel with the target model. When draft tokens are accepted, generation is faster. When rejected, compute is wasted.

If speculative decoding is enabled, testers MUST report:

Parameter	Description
Draft model	Identifier and parameter count
Speculation window (k)	Tokens proposed per verification step
Acceptance rate	Fraction of draft tokens accepted
Verification overhead	Latency when draft tokens are rejected

Table 8

Acceptance rate directly affects efficiency:

```
High acceptance (80%), k=5:
  Expected accepted per step = 4 tokens
  Verification passes per output token = 0.25

Low acceptance (30%), k=5:
  Expected accepted per step = 1.5 tokens
  Verification passes per output token = 0.67

Result: 2.7x more verification overhead at low acceptance
```

Results with speculative decoding MUST be labeled separately and include observed acceptance rate.

4.1.4.2. Chunked Prefill

Chunked prefill splits long prompts into smaller pieces, processing each chunk and potentially interleaving with decode iterations from other requests. This reduces head-of-line blocking but increases total prefill time for the chunked request.

If chunked prefill is enabled, testers MUST report:

* Chunk size in tokens
* Whether chunks interleave with other requests
* Impact on TTFT for long prompts

4.1.5. Primary Metrics

From [I-D.gaikwad-llm-benchmarking-terminology]:

* Time to First Token (TTFT)
* Inter-Token Latency (ITL)
* Time per Output Token (TPOT)
* Output Token Throughput

4.1.6. Secondary Metrics

* Request Throughput
* Queue Depth over time
* Queue Residence Time

* Prefill Latency (TTFT minus queue residence)

* Batch Utilization

4.1.7. Benchmarking Constraints

Request rate saturation differs from token saturation. A system might handle 2000 output tokens per second but only 50 requests per second if scheduling overhead dominates. Testers SHOULD measure both dimensions.

Mixed-length workloads increase tail latency under continuous batching. Short requests arriving behind long prefills experience head-of-line blocking. When workload includes high length variance, measure fairness: the ratio of actual latency to expected latency based on request size.

4.2. AI Gateway Profile

4.2.1. Definition and Concepts

An AI Gateway is a network-facing intermediary that virtualizes access to one or more Model Engines. Gateways handle cross-cutting concerns that do not belong in the inference engine itself.

Gateways perform several functions that affect latency:

Request Processing: TLS termination, authentication, schema validation, and protocol translation. These operations add fixed overhead per request.

Routing: Selection of backend engine based on load, capability, or policy. Intelligent routing (e.g., KV-cache-aware) adds decision latency but may reduce overall latency by improving cache hit rates.

Caching: Gateways may implement response caching. Traditional exact-match caching has limited utility for LLM traffic due to low query repetition. Semantic caching (matching similar queries) improves hit rates but introduces quality risk from approximate matches.

Admission Control: Rate limiting and quota enforcement. Under load, admission control adds queuing delay or rejects requests.

4.2.2. Boundary Specification

Included in SUT:

* TLS termination
* Authentication and authorization
* Schema validation and protocol translation
* Load balancing across engines or model replicas
* Semantic cache lookup and population
* Admission control and rate limiting
* Retry and fallback logic
* Response normalization

Excluded from SUT:

* Model inference computation (handled by downstream engine)
* Content inspection for safety (handled by AI Firewall)

4.2.3. Baseline Requirement

Gateway overhead is meaningful only relative to direct engine access. Gateway benchmarks MUST declare measurement type:

Measurement Type	What It Includes
Aggregate	Gateway processing plus downstream engine latency
Differential	Gateway overhead only, relative to direct engine access

Table 9

To measure differential latency:

1. Benchmark the Model Engine directly (baseline)
2. Benchmark through the Gateway to the same engine (same workload, same conditions)
3. Compute delta: Gateway_overhead = Gateway_TTFT - Engine_TTFT

Report both absolute values and delta.

4.2.4. Load Balancing Disclosure

Load balancing strategy affects tail latency. Testers MUST report:

Configuration	Options	Impact
Algorithm	Round-robin, least-connections, weighted, adaptive	Tail latency variance
Health checks	Interval, timeout, failure threshold	Failover speed
Sticky sessions	Enabled/disabled, key type	Cache locality
Retry policy	Max retries, backoff strategy	Failure handling

Table 10

For intelligent routing (KV-cache-aware, cost-optimized, latency-optimized):

* Routing signals used (queue depth, cache locality, model cost)

* Decision latency overhead

* Routing effectiveness (e.g., cache hit improvement from routing)

4.2.5. Multi-Model Gateway

Modern gateways route to multiple backend models based on capability, cost, or latency.

When gateway routes to heterogeneous backends, testers MUST report:

* Model selection logic: Rule-based, cost-optimized, capability-based

* Backend composition: List of models and their roles

* Fallback behavior: Conditions triggering model switching

Per-model metrics SHOULD be reported separately.

Cross-gateway comparison requires backend normalization. Comparing Gateway A (routing to GPT-4) against Gateway B (routing to Llama-70B) conflates gateway performance with model performance.

4.2.6. Semantic Cache

Semantic caching matches queries by meaning rather than exact text. A cache hit on "What is the capital of France?" might serve a response cached from "France's capital city?" This improves hit rates but risks serving inappropriate responses for queries that are similar but not equivalent.

Configuration Disclosure:

Parameter	Example	Why It Matters
Similarity threshold	Cosine >= 0.92	Lower threshold: more hits, more mismatches
Embedding model	text-embedding-3-small	Affects similarity quality
Cache capacity	100,000 entries	Hit rate ceiling
Eviction policy	LRU, frequency-based	Long-term hit rate
Cache scope	Global, per-tenant, per-user	Security and hit rate tradeoff
TTL	1 hour	Staleness vs hit rate

Table 11

Required Metrics:

Metric	Definition
Hit rate	Fraction of requests served from cache
Hit rate distribution	P50, P95, P99 of per-session hit rates
Latency on hit	TTFT when cache serves response
Latency on miss	TTFT when engine generates
Cache delta	Latency_miss minus Latency_hit
Mismatch rate	Fraction of hits where cached response was inappropriate

Table 12

Mismatch rate requires evaluation. Testers SHOULD disclose evaluation methodology (human review, automated comparison, or LLM-as-judge).

Session Definition: For per-session metrics, define what constitutes a session: requests sharing a session identifier, requests from the same user within a time window, or another definition. Testers MUST disclose session definition.

Staleness in RAG Systems: When semantic cache operates with a RAG system, cached responses may reference documents that have since been updated.

Parameter	Description
Index update frequency	How often RAG index refreshes
Cache TTL	Maximum age of cached entries
Staleness risk	Estimated fraction of stale cache hits

Table 13

Staleness risk estimate:

```
staleness_risk = (average_cache_age / index_update_interval) * corpus_change_rate
```

Benchmarking Constraints: Workload diversity determines hit rate. Testers MUST report:

* Number of distinct query clusters in workload
* Cache state at test start (cold or warm)
* Time until hit rate stabilizes

4.3. AI Firewall Profile

4.3.1. Definition and Concepts

An AI Firewall is a bidirectional security intermediary that inspects LLM inputs and outputs to detect and prevent policy violations.

Unlike traditional firewalls that examine packet headers or match byte patterns, AI Firewalls analyze semantic content. They must understand what a prompt is asking and what a response is saying. This requires ML models, making firewall latency fundamentally different from network firewall latency.

The firewall sits on the request path and adds latency to every request. The core tradeoff: more thorough inspection catches more threats but costs more time.

4.3.2. Boundary Specification

Included in SUT:

* Prompt analysis and classification
* Output content inspection
* Policy decision engine
* Block, allow, or modify actions

Excluded from SUT:

* Model inference (upstream or downstream)
* Network-layer firewalling (traditional WAF)
* Authentication (handled by gateway)

4.3.3. Enforcement Directions

AI Firewalls operate bidirectionally. Each direction addresses different threats.

Inbound Enforcement inspects user prompts before they reach the model:

Threat	Description
Direct prompt injection	User attempts to override system instructions
Indirect prompt injection	Malicious content in retrieved documents
Jailbreak attempts	Techniques to bypass model safety training
Context poisoning	Adversarial content to manipulate model behavior

Table 14

Outbound Enforcement inspects model outputs before they reach the user:

Threat	Description
PII leakage	Model outputs personal information
Policy violation	Output violates content policies
Tool misuse	Model attempts unauthorized actions
Data exfiltration	Sensitive information encoded in output

Table 15

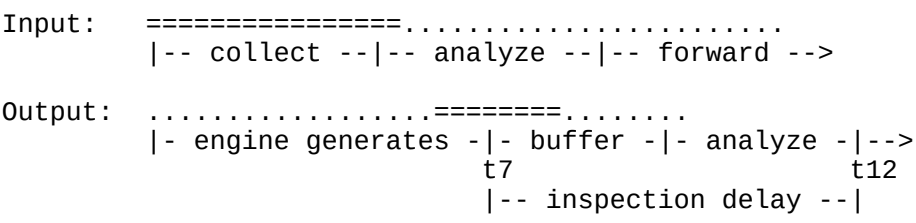
Testers MUST declare which directions are enforced. A benchmark testing inbound-only enforcement MUST NOT claim protection against outbound threats.

4.3.4. Inspection Architecture

Firewalls use different inspection strategies with distinct latency characteristics.

INSPECTION ARCHITECTURE COMPARISON

BUFFERED (adds to TTFT, no ITL impact):



STREAMING (no TTFT impact, adds to ITL):

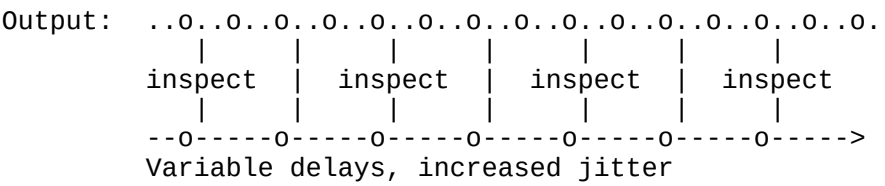


Figure 6: Inspection Architecture Comparison

Buffered Inspection: The firewall collects complete input (or output) before analysis.

Characteristics:

* Adds to TTFT (inbound) or delays token delivery (outbound)

* No impact on ITL once streaming starts

* Enables deep analysis requiring full context

For outbound buffered inspection, the client receives the first token later than the engine generates it. This distinction matters:

Engine TTFT (t7 - t6): 200ms
Outbound inspection: 50ms
Client-observed TTFT (t12 - t0): 250ms + network

Streaming Inspection: The firewall analyzes content as tokens flow through.

Characteristics:

* Adds per-token overhead to ITL

* May batch or pause tokens during analysis

* Introduces jitter in token delivery

Required measurements:

Metric	Definition
Per-token inspection delay	Average latency added per token
Maximum pause duration	Longest delay during streaming
Pause frequency	How often inspection causes batching
Jitter contribution	Standard deviation of delays

Table 16

Hybrid Inspection: Initial buffering followed by streaming. Common pattern: buffer first N tokens for context, then stream with spot-checks.

Configuration to disclose:

* Buffer threshold (tokens before streaming starts)

* Spot-check frequency

* Escalation triggers (patterns that switch to full buffering)

4.3.5. Required Metrics

Accuracy Metrics:

Metric	Definition
Detection Rate	Fraction of malicious inputs correctly blocked
False Positive Rate (FPR)	Fraction of benign inputs blocked by firewall
False Refusal Rate (FRR)	Fraction of policy-compliant requests refused at system boundary
Over-Defense Rate	FPR conditional on trigger-word presence in benign inputs

Table 17

FPR vs FRR: FPR measures firewall classifier errors on a benign test set. FRR measures all refusals observed at the system boundary, which may include:

* Firewall blocks (captured in FPR)

* Model refusals (model's own safety behavior)

* Policy engine blocks (business rules)

* Rate limiting (capacity rejection)

Therefore: FRR >= FPR when other refusal sources exist.

When reporting both, attribute refusals by source when possible.

Over-Defense Rate: Measures false positives on benign inputs that contain words commonly associated with attacks.

Over-Defense Rate = P(Block | Benign AND Contains_Trigger_Words)

Examples of benign inputs that may trigger over-defense:

* "Explain how prompt injection attacks work" (security education)

* "What does 'ignore previous instructions' mean?" (linguistic question)

* "How do I kill a process in Linux?" (technical query)

The test corpus for over-defense MUST contain semantically benign inputs that happen to include trigger words. Testing with trivially benign inputs does not measure real over-defense risk.

Latency Metrics:

Metric	Definition
Passing latency	Overhead when firewall allows request
Blocking latency	Time to reach block decision
Throughput degradation	Reduction in requests per second

Table 18

Latency may vary by decision path:

Example:
Allow (no flags): 8ms
Allow (flagged, deep analysis): 45ms
Block (pattern match): 3ms
Block (semantic analysis): 67ms

Report latency distribution by decision type.

4.3.6. Workload Specification

AI Firewall benchmarks require careful workload design.

Benign Workload: Normal traffic with no policy violations. Measures passing latency, FRR, and throughput impact on legitimate use. Source: Sanitized production samples or standard datasets.

Adversarial Workload: Known attack patterns. Measures detection rate, blocking latency, and FPR. Source: Published datasets (BIPIA [BIPIA], JailbreakBench, PromptInject) or red team generated. Do not publish working exploits.

Mixed Workload (recommended): Combines benign and adversarial at declared ratio.

Parameter	Example
Mix ratio	95% benign, 5% adversarial
Adversarial categories	40% injection, 30% jailbreak, 30% PII
Arrival pattern	Uniform or bursty

Table 19

4.3.7. Multi-Layer Firewall

Production deployments often stack multiple inspection layers.

```
Request -> Quick Filter -> ML Classifier -> Model -> Semantic Check -> PII Scan -> Respon
se
              |               |                          |              |
           regex           embedding                 output          entity
           + rules         classifier               analysis        detection
```

When multiple layers exist, report:

* Number and position of layers
* Per-layer latency
* Execution model: Series (latencies add) or parallel
* Short-circuit behavior: Does blocking at layer N skip later layers?

Delta decomposition:

```
Total overhead = Quick(2ms) + ML(12ms) + Semantic(34ms) + PII(8ms) = 56ms

With short-circuit on input block:
  Overhead = Quick(2ms) + ML(12ms) = 14ms
```

4.3.8. Benchmarking Constraints

Blocking speed alone is meaningless. A firewall blocking all requests in 1ms is useless. Always measure impact on benign traffic alongside detection effectiveness.

Disclose integration with WAF, rate limiting, or DDoS protection. These add latency.

Different attack categories have different detection latencies. Pattern-based detection is faster than semantic analysis. Report detection latency by category.

4.4. Compound System Profile

4.4.1. Definition and Concepts

A Compound System executes multiple inference, retrieval, and tool-use steps to satisfy a user intent. The system orchestrates these steps, manages state across them, and produces a final response.

Examples: RAG pipelines, multi-agent systems, tool-using assistants, coding agents.

Unlike single-inference benchmarks, compound system benchmarks measure task completion, not token generation. The primary question is "Did it accomplish the goal?" not "How fast did it generate tokens?"

4.4.2. Boundary Specification

Included in SUT:

* Orchestration and planning logic
* Multiple LLM inference calls
* Retrieval pipeline (embedding, search, reranking)
* Tool execution environment
* Conversation state management
* Agent-to-agent communication

Excluded from SUT:

* External APIs outside the system boundary (latency measured but not controlled)
* User interface rendering
* Arbitrary user-supplied code

Boundary Rule: The Compound System boundary includes only components deployed and controlled as part of the serving system. User-provided plugins or custom code at runtime are excluded. This prevents ambiguity when comparing systems with different extensibility models.

Component	Included?	Rationale
Built-in retrieval	Yes	Part of serving system
Standard tool library	Yes	Shipped with system
User-uploaded plugin	No	User-supplied
External API (weather)	Latency measured	Outside boundary

Table 20

4.4.3. Primary Metrics

Metric	Definition
Task Completion Latency	Time from user request to final response
Task Success Rate	Fraction of tasks completed correctly

Table 21

Task Success has two dimensions:

Type	Definition	Evaluation
Hard Success	Structural correctness	Automated (valid JSON, no errors)
Soft Success	Semantic correctness	Requires evaluation

Table 22

4.4.4. Evaluation Oracle

When using automated evaluation for Task Success Rate, disclose oracle methodology.

LLM-as-Judge:

Parameter	Report
Judge model	Identifier and version
Judge prompt	Full prompt or published rubric reference
Ground truth access	Whether judge sees reference answers
Sampling	Temperature, judgments per task

Table 23

Report inter-rater agreement if using multiple judges.

Rule-Based Evaluation:

Parameter	Report
Rule specification	Formal definition
Coverage	Fraction of criteria that are rule-checkable
Edge case handling	How ambiguous cases resolve

Table 24

Human Evaluation:

Parameter	Report
Evaluator count	Number of humans
Rubric	Criteria and scoring
Agreement	Inter-rater reliability (e.g., Cohen’s Kappa)
Blinding	Whether evaluators knew system identity

Table 25

4.4.5. Secondary Metrics

Metric	Definition
Trace Depth	Sequential steps in execution
Fan-out Factor	Maximum parallel sub-requests
Sub-Request Count	Total LLM calls per user request
Loop Incidence Rate	Fraction of tasks with repetitive non-progressing actions
Stalled Task Rate	Fraction of tasks hitting step limit without resolution
State Management Overhead	Latency and memory for multi-turn context

Table 26

Stalled Task Rate:

Stalled Task Rate = Tasks_reaching_max_steps / Total_tasks

Stalled tasks differ from loops. A loop repeats similar actions. A stalled task may try diverse actions but fail to converge. Both indicate problems but different ones.

4.4.6. RAG Sub-Profile

When Compound System includes Retrieval-Augmented Generation:

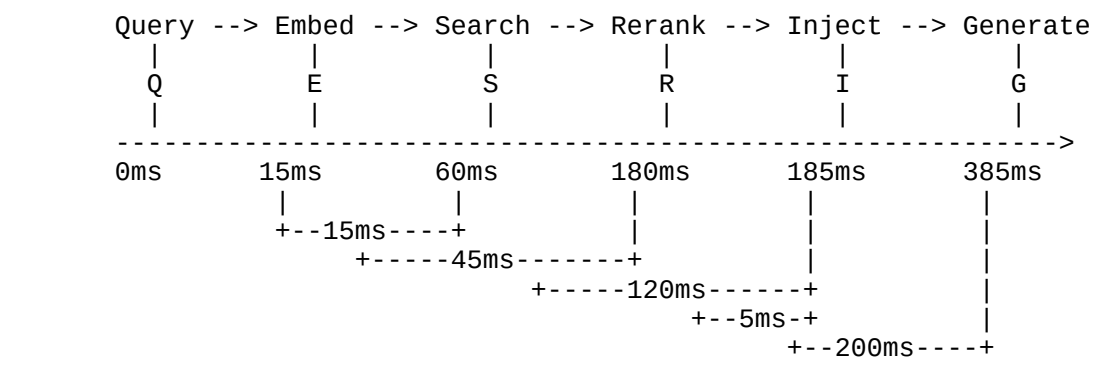


Figure 7: RAG Pipeline Latency

Configuration Disclosure:

Component	Parameters
Embedding	Model, dimensions, batch size
Vector store	Type, index configuration
Search	Top-k, similarity metric, filters
Reranking	Model (if used), top-n after rerank
Context	Max tokens, formatting template

Table 27

RAG-Specific Metrics:

Metric	Definition
Embedding Latency	Query to vector conversion
Retrieval Latency	Search and fetch time
Retrieval Recall	Fraction of relevant docs retrieved
Context Injection Overhead	Additional prefill from retrieved content

Table 28

Corpus Constraints:

Characteristic	Impact
Corpus size	Larger means longer search
Document length	Longer means more context overhead
Semantic diversity	More diverse reduces precision

Table 29

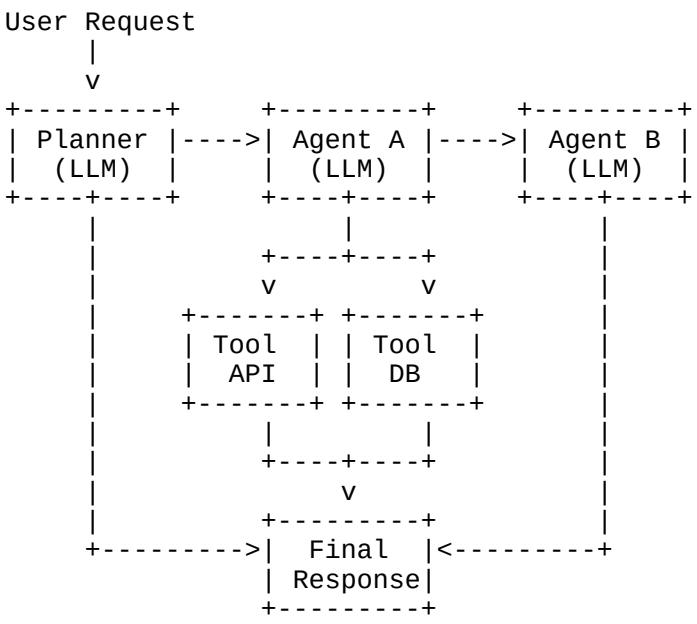
Report corpus statistics: document count, average length, domain.

Vector index must be fully built before measurement.

4.4.7. Agentic System Boundaries

For multi-agent or tool-using systems:

AGENTIC EXECUTION TRACE



Trace depth: 4 (Planner -> A -> Tools -> B)
Fan-out: 2 (parallel tool calls)
Sub-requests: 3 LLM calls

Figure 8: Agentic Execution Trace

Definitions:

Term	Definition
Agent invocation	Single LLM call with specific role
Tool call	External capability invocation
Orchestration step	Planning/routing decision
Trace	Complete sequence for one user request

Table 30

Measurement Points:

Metric	Start	End
Per-agent latency	Agent receives input	Agent produces output
Per-tool latency	Tool call initiated	Response received
Orchestration overhead	Previous step complete	Next step starts
Task completion	User request received	Final response delivered

Table 31

4.4.8. Exclusions

Custom user application logic and bespoke agent frameworks are out of scope. This profile covers general patterns, not specific implementations.

5. Workload Profiles

Workload profiles specify traffic patterns applied to infrastructure profiles. They do not define measurement boundaries.

5.1. Chatbot Workload Profile

5.1.1. Characteristics

Characteristic	Description
Interaction	Stateful, multi-turn
Delivery	Streaming
Arrival	Closed-loop (user thinks between turns)
Session length	Variable, typically 3-20 turns

Table 32

5.1.2. Required Parameters

Parameter	Description	Example
Arrival model	Open or closed loop	Closed-loop
Think-time	User delay between turns	Exponential, mean=5s
Input length	Tokens per user message	Log-normal, median=50
Output length	Tokens per response	Log-normal, median=150
Context retention	History handling	Sliding window, 4K tokens
Session length	Turns per conversation	Geometric, mean=8

Table 33

5.1.3. Example

Chatbot Workload: Customer Support

```
Arrival: Closed-loop, 100 concurrent sessions
Think-time: Exponential(mean=8s)
Input: Log-normal(mu=4.0, sigma=0.8), range [10, 500]
Output: Log-normal(mu=5.0, sigma=1.0), range [20, 1000]
Context: Sliding window, last 4000 tokens
Session: Geometric(p=0.12), mean ~8 turns
System prompt: 200 tokens, shared
```

5.2. Compound Workflow Workload Profile

5.2.1. Characteristics

Characteristic	Description
Execution	Multi-step, may include parallel branches
Tool usage	API calls, code execution, database queries
Dependencies	Steps may depend on previous outputs

Failure modes	Steps may fail, requiring retry or alternatives

Table 34

5.2.2. Required Parameters

Parameter	Description	Example
Task complexity	Steps per task	Fixed=5 or distribution
Fan-out pattern	Parallel vs sequential	Max parallel=3
Tool latency	External dependency behavior	Real, mocked, simulated
Failure injection	Simulated failures	5% tool failure rate
Retry behavior	Failure handling	Max 2 retries, exponential backoff

Table 35

5.2.3. External Dependency Handling

Compound workflows depend on external systems. Disclose handling:

Approach	Description	When
Real	Actual API calls	Production-representative
Mocked	Fixed responses	Controlled experiments
Simulated	Statistical model	Reproducible benchmarks

Table 36

Report observed latency and failure rate for real dependencies.
Report configured values for mocked dependencies.

6. Delta Measurement Model

Section 4 and Section 5 defined individual profiles. Production systems compose multiple profiles. A request may pass through Gateway, Firewall, and Engine before response generation.

Meaningful comparison across composed systems requires attributing latency to each component. This section defines the delta measurement model.

6.1. Timestamp Reference

Consider a request flowing through a full stack:

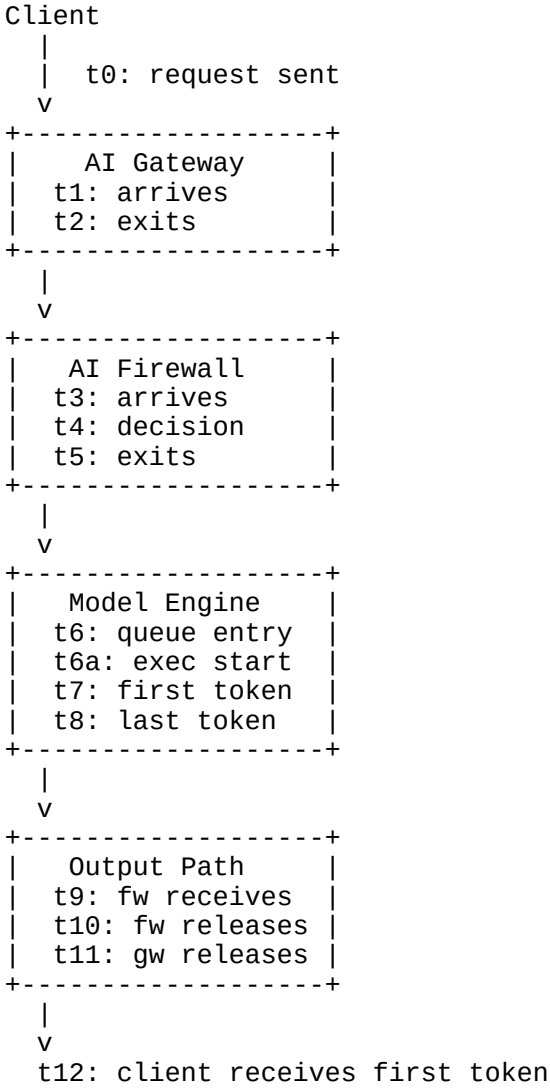


Figure 9: Request Flow Timestamps

6.2. Timestamp Definitions

Timestamp	Location	Event
t0	Client	Request transmission begins
t1	Gateway	Request arrives

t2	Gateway	Request exits toward firewall
t3	Firewall	Request arrives
t4	Firewall	Inbound decision reached
t5	Firewall	Request exits toward engine
t6	Engine	Request enters queue
t6a	Engine	Prefill computation begins
t7	Engine	First output token generated
t8	Engine	Last output token generated
t9	Firewall	First token arrives for outbound inspection
t10	Firewall	First token released after inspection
t11	Gateway	First token exits toward client
t12	Client	Client receives first token

Table 37

6.3. Component Deltas

Component	Formula	Measures
Gateway inbound	t2 - t1	Auth, validation, routing
Firewall inbound (pass)	t5 - t3	Prompt inspection
Firewall inbound (block)	t4 - t3	Time to block
Engine queue	t6a - t6	Wait before execution
Engine prefill	t7 - t6a	Prefill computation
Engine TTFT	t7 - t6	Queue plus prefill

Firewall outbound	t10 - t9	Output inspection
Gateway outbound	t11 - t10	Response processing

Table 38

6.4. End-to-End Metrics

Metric	Formula	Notes
Engine TTFT	t7 - t6	At engine boundary
System TTFT	t12 - t0	Client-observed
Output path overhead	t12 - t7	Delay from engine emit to client receive

Table 39

6.5. Clock Synchronization

Delta metrics within a single component (t2 - t1, both from gateway clock) are reliable. Cross-component deltas (t6 - t5) require clock synchronization.

For end-to-end metrics involving client timestamps (t0, t12), clock skew introduces error.

Options:

1. Single-machine measurement (client and server share clock)

2. Measure and report skew bounds

3. Report server-side metrics only when skew is too large

Recommended practice: Calculate deltas within components rather than across boundaries when possible.

See Section 9.3 for synchronization requirements.

7. Profile Composition

7.1. Composite SUT Declaration

When SUT includes multiple profiles, testers MUST:

1. Enumerate all components in request path:

```
Client -> AI Gateway -> AI Firewall -> Model Engine -> AI Firewall -> Client
```

2. Declare measurement boundary:

Type	Description
Full-stack	Client to response, all components
Per-component	Separate measurement at each boundary
Partial	Specific subset (e.g., Gateway + Engine)

Table 40

3. Provide delta decomposition:

Component	TTFT Contribution	Throughput Impact
AI Gateway	+15ms	-3%
AI Firewall (in)	+45ms	-8%
Model Engine	180ms (baseline)	baseline
AI Firewall (out)	+12ms*	-12%
Total	252ms	-22%

*Outbound adds to client-observed TTFT, not engine TTFT

7.2. Composition Validation

Measure components independently before measuring composite:

1. Engine alone: TTFT_engine = 180ms

2. Gateway + Engine: TTFT_gw = 195ms, Gateway_delta = 15ms

3. Firewall + Engine: TTFT_fw = 225ms, Firewall_delta = 45ms

4. Full stack: TTFT_full = 252ms

5. Validate: TTFT_engine + deltas approximately equals TTFT_full

If validation fails, interaction effects exist. Document them.

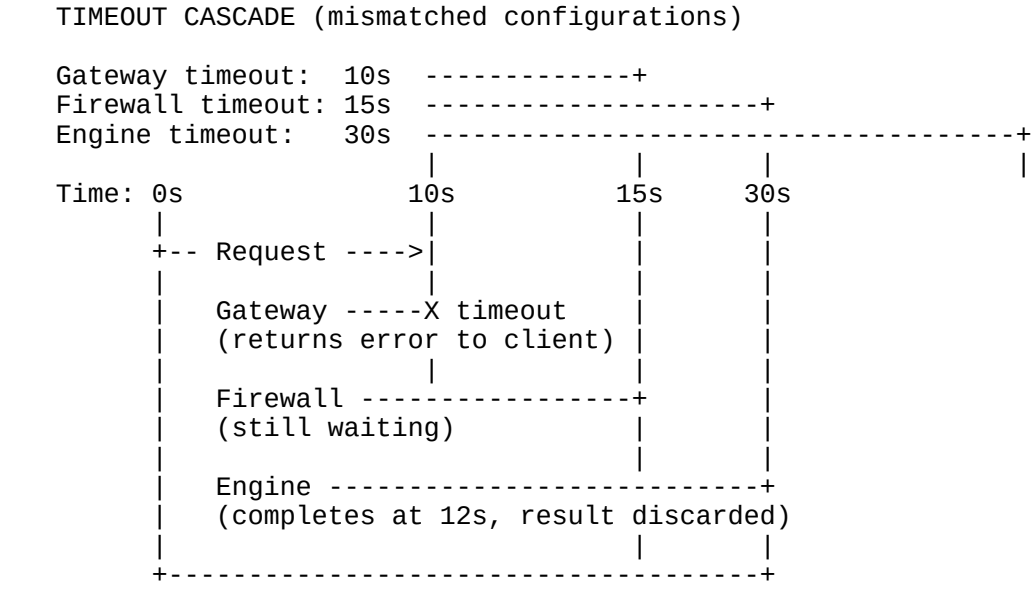
7.3. Interaction Effects

Components may interact beyond simple addition:

Effect	Description	Example
Batching interference	Gateway batching conflicts with engine	Gateway batches 8, engine max is 4
Cache interaction	High gateway cache hit means engine sees hard queries	Biased difficulty
Backpressure	Slow component causes upstream queuing	Firewall slowdown grows gateway queue
Timeout cascades	Mismatched timeouts waste resources	See below

Table 41

Timeout Cascades:



Result: Client gets error at 10s. Engine wastes 12s of compute.

Figure 10: Timeout Cascade Example

Report timeout configurations and note mismatches.

8. Access Logging Requirements

8.1. Minimum Fields

All profiles MUST log:

Field	Description
timestamp	Request start time
request_id	Unique identifier
profile	Infrastructure profile under test
workload	Workload profile applied
latency_ms	Total request latency
status	Success, error, timeout

Table 42

8.2. Model Engine Fields

Field	Description
queue_time_ms	Time in queue
prefill_time_ms	Prefill latency
decode_time_ms	Generation time
batch_size	Concurrent requests in batch
token_count_in	Input tokens
token_count_out	Output tokens

Table 43

8.3. AI Firewall Fields

Field	Description
direction	Inbound or outbound
decision	Allow, block, modify
policy_triggered	Which policy matched

confidence	Detection confidence
inspection_time_ms	Analysis time

Table 44

8.4. Compound System Fields

Field	Description
trace_id	Identifier linking all steps
step_count	Total orchestration steps
tool_calls	List of tools invoked
success_type	Hard, soft, or failure

Table 45

8.5. AI Gateway Fields

Field	Description
cache_status	Hit, miss, or bypass
route_target	Selected backend
token_count_in	Input tokens
token_count_out	Output tokens

Table 46

8.6. OpenTelemetry Integration

OpenTelemetry integration SHOULD be supported. Reference GenAI semantic conventions when available.

9. Measurement Considerations

9.1. Baseline and Delta Reporting

For intermediary components (Gateway, Firewall), provide differential measurements:

1. Measure downstream directly (baseline)
2. Measure through intermediary
3. Compute delta
4. Report both absolute and delta

9.2. Warm-up and Steady State

Declare whether results include cold start.

Profile	Cold Start Factors
Model Engine	JIT compilation, KV cache allocation, batch ramp-up
AI Gateway	Connection pool, cache population
AI Firewall	Model loading, rule compilation
Compound System	All above plus retrieval index loading

Table 47

If excluding cold start, report warm-up procedure and duration.

9.3. Clock Synchronization

Configuration	Minimum Accuracy	Method
Single-machine	Inherent	N/A
Same rack	1ms	NTP

Distributed	100us	PTP
Sub-millisecond analysis	10us	PTP with hardware timestamps

Table 48

Reports MUST declare:

* Synchronization method

* Estimated maximum skew

* Single-point or distributed measurement

9.4. Streaming Protocol Considerations

Profile	Recommended Protocol	Notes
Model Engine	gRPC streaming	Lower overhead
AI Gateway	SSE over HTTP	Broad compatibility
AI Firewall	Match upstream/downstream	Minimize translation
Compound System	SSE or WebSocket	Client dependent

Table 49

Report chunk size distribution when measuring ITL.

10. Security Considerations

10.1. Bidirectional Enforcement Gaps

AI Firewalls enforcing only one direction leave systems exposed.

Inbound-only gaps:

* Cannot prevent PII leakage

* Cannot catch policy violations from model

* Cannot stop tool misuse

Outbound-only gaps:

* Cannot prevent prompt injection

* Cannot stop jailbreak attempts

* Malicious content reaches model

Declare which directions are enforced. "AI Firewall protection" without direction is incomplete.

10.2. Adversarial Workload Handling

Security requirements for adversarial benchmarks:

* Samples MUST NOT contain working exploits

* Use sanitized patterns or synthetic constructs

* Reference published taxonomies (OWASP LLM Top 10, MITRE ATLAS)

* Do not publish novel attacks discovered during testing

10.3. Side-Channel Considerations

Performance characteristics may leak information:

Channel	Risk	Mitigation
Timing	Decision time reveals classification	Add noise
Cache	Hit patterns reveal similarity	Per-tenant isolation
Routing	Balancing reveals backend state	Randomize

Table 50

Multi-tenant benchmarks SHOULD measure side-channel exposure.

11. References

11.1. Normative References

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

[I-D.gaikwad-llm-benchmarking-methodology]
Gaikwad, M., "Benchmarking Methodology for Large Language Model Serving", Work in Progress, Internet-Draft, draft-gaikwad-llm-benchmarking-methodology, 2026, <https://datatracker.ietf.org/doc/html/draft-gaikwad-llm-benchmarking-methodology>.

[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>.

11.2. Informative References

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

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

[OWASP-LLM]
OWASP Foundation, "OWASP Top 10 for Large Language Model Applications", 2023, <https://owasp.org/www-project-top-10-for-large-language-model-applications/>.

[BIPIA] Yi, J., Xie, Y., Zhu, B., Hines, K., Kiciman, E., Sun, G., and X. Xie, "Benchmarking and Defending Against Indirect Prompt Injection Attacks on Large Language Models", 2023, <https://arxiv.org/abs/2312.14197>.

[DISTSERVE]
Zhong, Y., Liu, S., Chen, J., Hu, J., Zhu, Y., Liu, X., Jin, X., and H. Zhang, "DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model

Serving", OSDI 2024, 2024,
<https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin>.

[MOONCAKE] Qin, R., Li, Z., He, W., Zhang, M., Wu, Y., Zheng, W., and L. Zhou, "Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving", 2024,
<https://arxiv.org/abs/2407.00079>.

Appendix A. Example Benchmark Report Structure

1. Executive Summary
   - SUT and profile(s) used
   - Key results

2. System Configuration
   - Hardware
   - Software versions
   - Profile-specific config (per Section 4)

3. Workload Specification
   - Workload profile
   - Parameters (per Section 5)
   - Dataset sources

4. Methodology
   - Measurement boundary
   - Clock synchronization
   - Warm-up procedure
   - Duration and request counts

5. Results
   - Primary metrics with percentiles
   - Secondary metrics
   - Delta decomposition (if composite)

6. Analysis
   - Observations
   - Interaction effects
   - Limitations

7. Reproduction
   - Config files
   - Scripts
   - Random seeds

Author’s Address

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