vmware by Broadcom

Gateway to Enterprise AI

Architecting and Scaling LLM Workloads with Envoy

White Paper

Table of Contents

Chapter 1: Abstract and Introduction

The enterprise adoption of large language models (LLMs) presents significant architectural challenges that extend far beyond simply hosting a model. Organizations demand robust, high-performance AI gateways capable of providing identity-based routing, strict rate limiting, zero-trust security, and granular observability. Managing LLM traffic is uniquely demanding because it typically involves long-lived, chunked HTTP/2 streams that generate intense memory and connection pressure. Furthermore, establishing authoritative FinOps controls for chargeback and ensuring secure credential management are critical prerequisites for enterprise integration. Deploying a traditional AI gateway in front of generative AI engines introduces the significant risk of a *streaming tax*, which is the latency overhead imposed by the proxy when buffering and processing these large, continuous token streams.

To address these architectural requirements, we engineered a comprehensive validation framework to evaluate the Envoy AI Gateway deployed on a VMware® Kubernetes Service cluster running on top of VMware Cloud Foundation® (VCF). This document serves as a best practices guide for VCF enterprise customers seeking to evaluate, architect, and deploy high-performance AI gateway solutions on VMware Kubernetes Service. In our validation environment, the gateway was deployed as the central control plane to seamlessly route traffic between two distinct performance tiers:

- Tier 1 (premium/frontier) routes to an external cloud model designed exclusively for highly complex, reasoning-heavy agentic tasks.
- Tier 2 routes to an on-premises vLLM inference engine serving a large 120 billion parameter open-source model (`gpt-oss-120b`).

The infrastructure was rigorously designed to ensure that enterprise policies (such as tenant isolation, token-aware rate limiting, and zero-trust credential injection) could be enforced at the endpoint without degrading the downstream performance of the underlying high-performance hardware.

The primary objective of this validation initiative was to establish the performance, scalability, and policy-routing capability of the Envoy AI Gateway under configured policies. By simulating a complex mix of unpredictable human interactions and intense, highly correlated agentic bursts, we aimed to empirically evaluate the gateway's routing efficiency and quantify the exact latency overhead introduced by the proxy. The central thesis of this document is to show that with meticulous architectural design and strict policy alignment, the Envoy AI Gateway can properly manage aggressive, mixed agentic and human workloads without violating the strict service-level objectives (SLOs) required for interactive human-paced consumption.

Chapter 2: Infrastructure and Platform Architecture

Our architecture centers on the deployment of the Envoy AI Gateway on VMware Kubernetes Service, strategically positioned to steer and throttle traffic destined for a backend vLLM cluster powered by four interconnected NVIDIA H100 graphics processing units (GPUs). The VMware Kubernetes Service environment acts as the enterprise Kubernetes runtime, running on top of VCF to deliver a highly integrated, reliable private cloud substrate.

By leveraging VCF as the foundational private cloud infrastructure, enterprise operators benefit from highly optimized compute, networking, and storage virtualization. The VMware Kubernetes Service environment running on VCF provides a robust, scalable Kubernetes substrate with deep hypervisor integration. At the same time, the multi-GPU backend leverages Tensor Parallelism (TP=4) across high-speed NVLink connections to distribute the large memory requirements of the 120B parameter model. This decoupling of the control plane (Envoy) from the data plane (vLLM) is essential for maintaining stability, as it allows the gateway to absorb connection bursts and shed illegitimate load before it ever reaches the highly constrained GPU compute layer, presenting an architectural gold standard for VCF private cloud deployments.

The foundation of the gateway's traffic management strategy relies significantly on Envoy's Custom Resource Definition (CRD) native capabilities, enabling native, policy-driven control at the endpoint:

- `AIGatewayRoute` (identity-based routing): Unlike traditional URL-based routing, the `AIGatewayRoute` CRD implements sophisticated identity-based routing. The gateway inspects inbound payloads for the `x-workload-app` HTTP header to definitively identify the calling application, such as an automated `toolbench-agent` or a human-facing `swe-rebench-ide`, and dynamically routes traffic across two distinct performance tiers:
  - Tier 1 (premium/frontier): Routes to an external cloud model (for example, Gemini Flash) reserved exclusively for highly complex, reasoning-heavy agentic tasks.
  - Tier 2 (standard/cost-effective): Routes to our on-premises vLLM cluster serving `gpt-oss-120b`, acting as a *cheap*, high-throughput workhorse for standard background operations. This architecture provides significant FinOps and business value; not every LLM request requires the immense cost of a frontier model. By actively inspecting the identity tag at the endpoint, the Envoy Gateway acts as a powerful cost-optimizer. It intelligently offloads large, repetitive workloads to the effectively *free* on-premises GPUs, while preserving the expensive cloud API budget strictly for applications that genuinely demand frontier-level intelligence.
- `BackendSecurityPolicy` (zero-trust security): To secure this multi-tenant architecture, the `BackendSecurityPolicy` CRD enforces an uncompromising zero-trust authentication model. Enterprise security mandates that edge clients must never possess direct access to upstream backend API keys, nor should they be trusted if they supply them. The `BackendSecurityPolicy` is configured to intercept all inbound traffic, proactively strip any client-provided authorization headers, and securely inject the genuine API keys retrieved dynamically from Kubernetes Secrets. This ensures that only authenticated workloads, verified by the gateway, can access the premium GPU resources.
- `BackendTrafficPolicy` (token-aware rate limiting): Finally, to prevent any single rogue agent from monopolizing the limited Key-Value (KV) cache Video RAM (VRAM) of the GPU cluster, the `BackendTrafficPolicy` CRD establishes token-aware local rate limiting. By defining precise local rate limits based on client selectors (matching the `x-workload-app` header), the gateway enforces strict token and request budgets for different workloads. This policy operates directly at the edge, proactively shedding excess load with `HTTP 429 Too Many Requests` responses. This defense mechanism is absolutely critical, as it guarantees that sudden, explosive parallel bursts from autonomous multi-agent systems do not trigger catastrophic out-of-memory (OOM) crashes on the vLLM backend.

Chapter 3: Observability and FinOps Strategy

In a production-grade AI gateway deployment, observability cannot be treated as a monolithic concept; it requires a highly federated approach. Our architecture strictly separates the following data:

- Gateway telemetry: Focusing on routing decisions, rate limiting, and authoritative token counting at the endpoint.
- Backend telemetry: Focusing on hardware utilization, KV cache states, and inference phase latencies.

To achieve a holistic, zero-blind spot view, our Prometheus stack was explicitly configured for tri-target scraping, actively polling the following components:

1. Envoy AI Gateway endpoint on VMware Kubernetes Service to monitor proxy health and routing performance.
2. vLLM Inference Engine endpoint to capture detailed generation phase histograms, such as time-to-first-token (TTFT) and time-per-output-token (TPOT).
3. NVIDIA Data Center GPU Manager (DCGM) Exporter on the backend node to measure physical GPU metrics such as VRAM saturation and NVLink cross-communication bandwidth.

A foundational component of the enterprise integration was establishing an authoritative FinOps and chargeback model. In a zero-trust enterprise environment, the backend LLM engines cannot be trusted to report their own usage for billing purposes. Instead, the Envoy AI Gateway must serve as the single, immutable source of truth.

We utilized Envoy's ExtProc tokenizer to calculate token consumption directly at the proxy layer, which exposes the `gen_ai_client_token_usage_token_sum` metric:

- `gen_ai_client_token_usage_token_sum`: This specific metric represents the cumulative sum of all tokens (both prompt and generation) consumed by the client, sourced directly from the Envoy Gateway's Prometheus endpoint (`/stats/prometheus`).
- `gen_ai_request_model`: To enable granular cost allocation, the metric is natively aggregated by this label, which identifies the precise upstream AI model (for example, `gpt-oss-120b` or `gemini-flash`) that serviced the request.

By extracting these metrics from Envoy's endpoint, operators can multiply the token sum by tiered organizational rates to calculate the official chargeback costs for different business units.

However, operating independent tokenizers at the gateway and the backend introduces the risk of discrepancies. Different tokenizer architectures, or even different versions of the same tokenizer, can yield slightly different token counts for the exact same payload. To safeguard against this, we implemented continuous auditing for *token drift*.

By mathematically comparing the ExtProc token count at the gateway against the prompt and generation counts reported by the vLLM backend, we actively measured the percentage of drift. Maintaining strict fidelity between these two layers is critical; high token drift indicates a systemic tokenizer mismatch that inevitably leads to enterprise revenue leakage or the inaccurate overcharging of downstream customers.

Chapter 4: Data Engineering and Workload Modeling

To accurately and aggressively stress the performance boundaries of the system, our Data Engineering Extract, Transform, Load (ETL) pipeline was tasked with synthesizing a large, highly complex corpus of traffic. The pipeline unrolled vast multi-turn conversational trajectories offline, resulting in an enormous 124-GB high-cardinality dataset explicitly engineered to contain over 20,000 unique interaction sessions drawn directly from five distinct open-source enterprise profiles:

- Profile A (API orchestration): Sourced from `ToolBench-v1`, contributing 5000 unique sessions.
- Profile B (terminal siege / CI/CD pipelines): Sourced from `Terminal-Bench 2.0`, representing automated continuous integration/continuous deployment logging tasks (identified by the `terminal-bench-ci` workload header) and contributing 865 unique sessions.
- Profile C (UI navigation / multimodal): Sourced from `OmniACT`, contributing 5000 unique sessions.
- Profile D (coding and developer assistance): Sourced from `SWE-rebench` and `CoderForge`, contributing an expansive 10,000 unique sessions combined (5000 each).

This extreme scale and high cardinality were not arbitrary; they were absolutely mandatory to validate the true limits of the compute layer. Testing with low-cardinality data, such as repeatedly looping through the same ten user sessions, creates a devastating architectural trap:

- Under large concurrency, identical repeated prompts cause the vLLM prefix caching mechanism to achieve an artificial hit rate approaching 99.9%.
- When the cache hit rate is artificially inflated, the engine effortlessly bypasses the extraordinarily expensive prefill compute phase.
- This entirely masks the true processing bottlenecks, VRAM fragmentation, and KV cache allocation thrashing that inevitably occur under authentic, highly variable production load.

Equally critical to the success of the load generation was the implementation of a *zero-CPU parsing guarantee* during the dataset construction phase. In a high-concurrency load test simulating hundreds of concurrent users, forcing the load generator workers to perform CPU-intensive tasks (such as dynamically parsing large JSON objects, managing sliding context windows, estimating token counts, or interpolating strings) inevitably causes the workers themselves to bottleneck.

This client-side CPU exhaustion artificially caps the request throughput before the gateway is ever fully stressed. By unrolling the conversations offline into stateful, standalone HTTP payloads (`.jsonl`) enclosed in a thin metadata envelope, we eliminated JSON serialization overhead at runtime. The Locust workers simply read raw strings and fired bytes over the wire, allowing the test harness to effortlessly saturate the Envoy Gateway.

During the ETL transformation phase, we encountered and mitigated a significant schema validation trap that threatened to invalidate the testing suite:

- Older datasets, particularly `ToolBench-v1`, utilized the deprecated OpenAI message format specifying `{"role": "function"}` to represent tool outputs in the conversational trajectory.
- Modern vLLM engines enforce incredibly strict Pydantic schema validation on all incoming requests. When the Envoy Gateway forwarded these legacy payloads to the backend, vLLM instantly rejected them with HTTP 400 Bad Request errors, causing the load tests to fail outright.
- The data pipeline systematically bypassed this by gracefully intercepting any `{"role": "function"}` declaration and mapping it to `{"role": "user"}`, perfectly preserving the payload's specific token geometry, size, and KV cache pressure while ensuring strict compliance with modern engine schemas.

4.1 The Agent Simulation Architecture: Offline Unrolling versus Live Execution

To address concerns regarding the validity of simulating autonomous multi-agent workloads without executing live agent runtimes during the load tests, we must analyze the behavioral, computational, and stateful equivalence of our offline simulation design.

In a live production environment, an autonomous agent orchestrator (built on frameworks such as LangChain or AutoGen) runs a stateful execution loop:

1. The agent generates a prompt.
2. Transmits it to the model through the gateway.
3. Parses the streaming output chunk by chunk.
4. Evaluates local execution logic or triggers external tool API calls.
5. Appends the tool results to the conversation history, and iterates.

While deploying hundreds of live agents in a testing cluster might seem ideal, it introduces immense client-side CPU bottlenecks due to concurrent JSON parsing, context accumulation, and orchestrator framework overhead. This client-side resource exhaustion artificially caps load generation throughput, masking the true saturation boundaries of the gateway and backend models.

Our simulation architecture resolves this scaling limitation by decoupling conversational state calculation from runtime execution through an offline unrolling pipeline. The ETL pipeline precomputes every single conversational turn across over twenty thousand sessions, encoding them into a flat series of sequential, preserialized HTTP requests stored in newline-delimited JSON format.

During the stress test, the Locust load generator workers do not run any complex logic, nor do they dynamically parse model outputs. Instead, they act as high-performance replay engines:

- Each worker reads the thin outer envelope metadata of a precomputed turn.
- Pauses for the exact, empirically recorded think-time to simulate cognitive processing and network delay.
- Forwards the preserialized payload bytes directly to the gateway.

This process shifts 100% of the state management and logic overhead offline, ensuring the load generator can easily saturate the network and gateway layers.

The flow of data and control between a live agent loop and our high-performance offline unrolling simulation is contrasted in the following figures.

Figure 1: Live Agent Loop: Stateful and CPU-Intensive

Figure 2: Offline Simulation Loop: Our Decoupled Architecture

From the perspective of the backend vLLM inference engine, this unrolled replay is computationally and behaviorally indistinguishable from a live, dynamic agent swarm. An LLM serving engine is entirely agnostic to the intelligence or statefulness of the client; it only processes the raw token geometry of incoming requests.

The vLLM scheduler and PagedAttention memory manager are exposed to the exact same environment:

- Prompt lengths and prompt content
- Generation request parameters
- Staggered arrival timings

Because the Locust workers replay the multi-turn conversations turn by turn with precise, staggered think times, they reconstruct the identical KV cache context accumulation, prefix caching hit rates, VRAM fragmentation patterns, and prefill/decoding compute pressure that a cluster of actual live agents would generate.

Similarly, from the perspective of the Envoy AI Gateway situated in the middle of the traffic flow, the simulation is completely transparent and functionally identical to a live deployment. The gateway operates at the HTTP and token levels, evaluating requests based on metadata and raw byte streams.

Because our Locust workers inject the exact application-identifying `x-workload-app` headers and present the identical payload bytes as actual applications, Envoy applies the identical information:

- Identity-based routing
- zero-trust credential injection
- Token-aware rate limiting policies

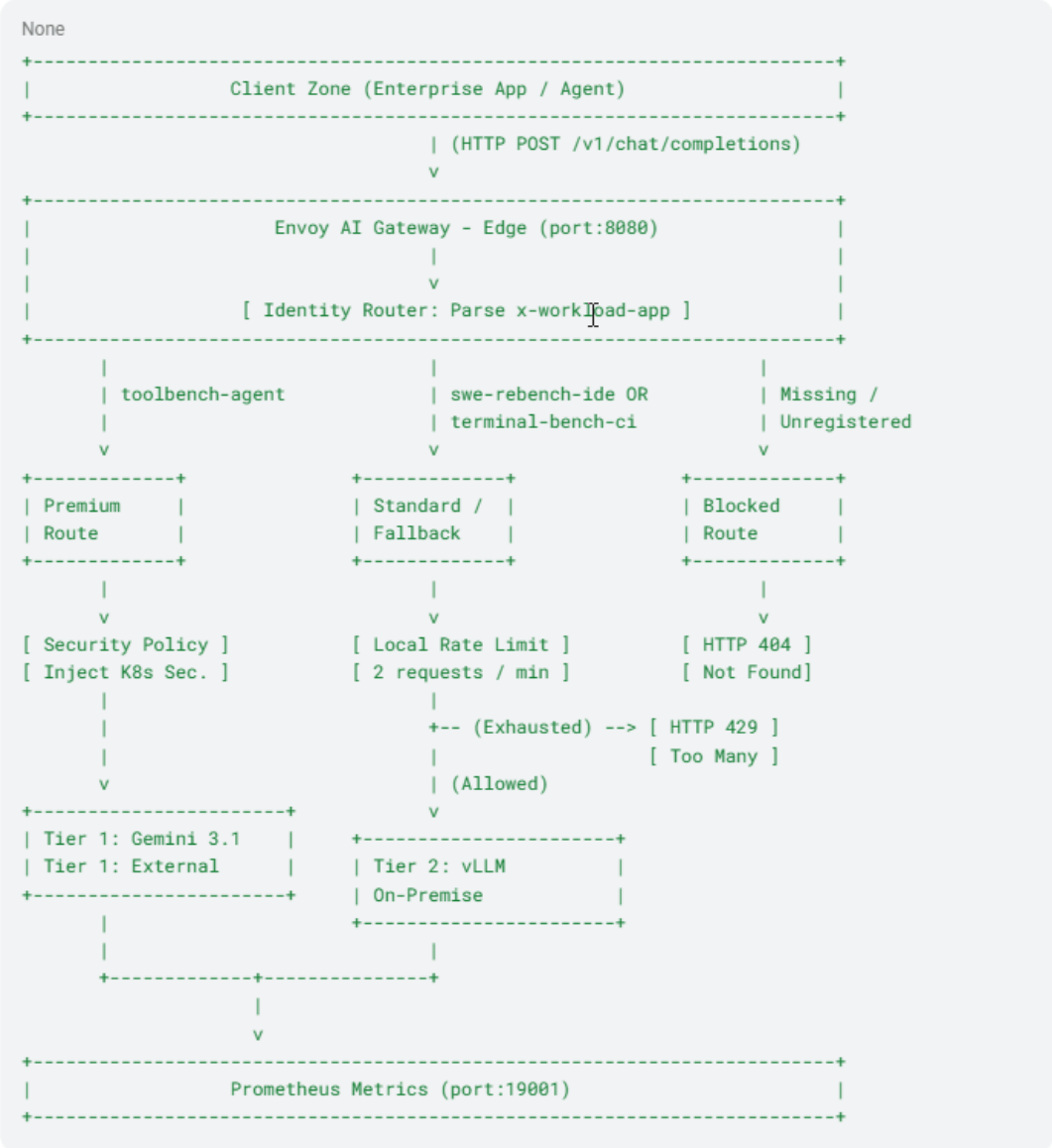
The proxy processes the exact same connection multiplexing, downstream buffering, and upstream connection pooling dynamics. Consequently, the isolated 2-millisecond gateway overhead documented in this study is a high-fidelity representation of Envoy's performance when mediating actual, live enterprise agent swarms.

Chapter 5: Functional Validation: Security and Routing

Before executing large, high-concurrency performance load tests, it was imperative to programmatically validate Envoy's core routing and security capabilities in an isolated, functional suite. Validating these invariants in a sterile environment guarantees that any failures observed during the stress testing phase are strictly related to compute saturation or network exhaustion, rather than fundamental misconfigurations of the edge proxies.

The following figure shows how the gateway parses identity headers to route specific workload profiles. For example, automated CI/CD pipelines utilizing the `terminal-bench-ci` header are directed to the standard fallback route, whereas complex agentic orchestrators are escalated to the premium route. Missing or unregistered identities are immediately blocked.

Figure 3: Routing Architecture Flow



5.1 Test Case 1: Local Rate Limiting

Objective: Validate Envoy's ability to protect the downstream vLLM infrastructure from catastrophic request floods while simultaneously enforcing strict FinOps cost controls.

Execution method: An Envoy `BackendTrafficPolicy` was configured with a strict local limit permitting exactly two requests per minute for any traffic presenting the `x-workload-app: swe-rebench-ide` header. A testing script systematically fired six sequential, rapid requests to intentionally exhaust the permitted local rate limit budget.

Actual results and evidence: SUCCESS. The initial two requests, falling within the established budget, successfully penetrated the gateway and returned `HTTP 200 OK` responses from the backend. Crucially, the gateway aggressively and instantaneously rejected the subsequent four requests, returning `HTTP 429 Too Many Requests`. Furthermore, querying the Envoy Prometheus metrics endpoint (`/stats/prometheus`) confirmed that the `http_local_rate_limit_rate_limited` counter accurately incremented exactly four times. This specific metric represents the total number of requests that were actively rejected by the gateway because they exceeded the defined token or request quota. Monitoring this metric provides the required active observability, definitively proving that the gateway is actively tripping limits and shedding load at the endpoint before it can impact the backend cluster.

5.2 Test Case 2: Zero-Trust Security

Objective: Validate the gateway's responsibility to securely manage external credential injection through Kubernetes Secrets, ensuring that edge developers never possess raw API keys.

Execution method: Initiated a request identifying as `x-workload-app: toolbench-agent` (a Tier 1 identity requiring external access), but the test payload was deliberately constructed to include a rogue `Authorization: Bearer malicious-token` HTTP header to simulate a compromised client attempting unauthorized access.

Actual Results and evidence: SUCCESS. The Envoy `BackendSecurityPolicy` operated flawlessly under these conditions. The gateway intercepted the inbound traffic, completely stripped the rogue `malicious-token` from the header payload, and dynamically fetched the genuine `gemini-api-key` stored safely within the cluster's Kubernetes secrets. The proxy securely injected the valid token and successfully routed the sanitized, fully authenticated request upstream, ultimately returning a valid HTTP 200 OK containing a response from the external LLM provider. Follow-up direct bypass attempts to the endpoint without the gateway's mediation failed entirely with HTTP 400 authentication errors, definitively proving the Gateway's capability to safely and exclusively mediate external AI access.

5.3 Test Case 3: Identity-Based Routing

Objective: Validate the gateway's core routing logic, confirming that tenant isolation and vendor independence could be achieved seamlessly by inspecting specific HTTP headers.

Execution method: Generated three perfectly identical JSON payload bodies targeting the exact same gateway endpoint, altering solely the `x-workload-app` HTTP header across the requests (`swe-rebench-ide, toolbench-agent`, and `unknown-guest`).

Actual results and evidence: SUCCESS. The gateway accurately parsed each header and executed distinct, correct routing decisions based on identity:

- The `swe-rebench-ide` request was correctly identified and routed to the cost-effective on-premises vLLM cluster (Tier 2) to save cloud costs on standard workloads, where it correctly encountered the local rate limit previously exhausted in Test Case 1, returning an HTTP 429.

- Conversely, the `toolbench-agent` request was successfully identified as a premium workload demanding advanced reasoning capabilities, and was intentionally routed to the frontier Gemini Flash model (Tier 1), successfully returning an HTTP 200 OK.
- Finally, traffic identifying as `unknown-guest` was immediately intercepted and rejected directly at the endpoint proxy with an HTTP 404 Not Found. This explicitly proved that unregistered, unverified, or malicious identities are aggressively blocked before they can consume any backend compute resources.

Chapter 6: Performance Test Methodology

Traditional performance testing methodologies, particularly those relying on static think times or smooth request intervals, fail entirely to capture the chaotic, unpredictable nature of LLM workloads. Static delays tend to create an artificial, synchronized pulsing effect where requests hit the gateway in predictable, uniform waves. This inherently flawed approach artificially smooths the aggregate load, completely failing to stress the first come, first served (FCFS) scheduler of the vLLM engine.

Real-world enterprise traffic is aggressively heterogeneous, characterized by wild, instantaneous spikes of activity followed by long periods of latency. To successfully discover the breaking point of the cluster, we abandoned static delays entirely.

To authentically simulate this heterogeneity, we engineered our tests using specific mathematical arrival distributions, each meticulously mapped to established queuing theory models to represent distinct traffic behaviors. Rather than choosing these distributions arbitrarily, we selected and parameterized our arrival distributions based on recent peer-reviewed systems research and empirical industry reports detailing production LLM gateway profiles:

- Markov-Modulated Poisson Processes (MMPP): Utilized for Profile A (API orchestration). Grounded in the structural analysis of multi-turn agentic environments (such as Microsoft's SOSP 2025 Pythia framework), autonomous agents do not trigger independent Poisson requests. Instead, tool execution graphs create causal, cascaded chains where arrival rates rely on prior execution latency. MMPP is an established theoretical model specifically designed for handling highly correlated, state-dependent arrivals. In our test, the Two-State ON/OFF Burst Generator used MMPP to simulate the behavior of autonomous agents initiating large parallel tool calls. It aggressively toggles between extreme burst states featuring near-zero interarrival delays and completely dormant states, perfectly mimicking the intense, highly correlated logic cycles of an agentic swarm rapidly consuming and processing external APIs.
- Gamma distribution (CV=1.5): Utilized for Profile B (Terminal Siege and CI/CD logging). Empirical production telemetry, specifically from Microsoft's BurstGPT dataset summarizing 10.31M Azure OpenAI traces, and the Alibaba Cloud ServeGen (2025) production LLM serving document, has shown that real-world LLM request patterns are heavily skewed. The ServeGen dataset characterization confirmed that interarrival times exhibit high coefficients of variation (typically exceeding 1.0), which completely refutes simplistic uniform or standard exponential Poisson assumptions. By modeling Profile B using a Gamma distribution with a high coefficient of variation (CV=1.5), we vigorously tested the system's ability to handle sudden, quadratic context growth. CI/CD pipelines and terminal logs frequently dump large, unstructured blocks of text into a conversational context at highly irregular intervals. The high variance of the Gamma distribution ensured that these large text block appends hit the gateway unpredictably, maximizing KV cache thrashing and forcing the PagedAttention allocators to rapidly re-evaluate contiguous block availability.
- Exponential distributions: Utilized for Profiles C and D (UI navigation and knowledge workers). Grounded deeply in classical queuing theory (such as the landmark Paxson and Floyd 1994 Internet traffic studies refuting simplistic Poisson models for complex sessions), exponential interarrival times represent the established mathematical gold standard for modeling independent, uncoordinated actors. This distribution accurately simulates uncoordinated human pacing, reflecting the reality that human knowledge workers read, pause, and interact with user interfaces independently of one another.

The interplay of these three distinct mathematical models created a chaotic, highly realistic load signature that genuinely pushed the limits of the hardware.

Rather than blindly stepping up concurrency at fixed time intervals, the overall execution was governed by a telemetry-gated saturation ramp. Load generation dynamically advanced only when the system proved mathematically stable. The ramp was programmed to discover two critical saturation points:

- T2 saturation point (absolute compute boundary): Triggered the moment the rolling average TTFT degraded superlinearly, exceeding 3.0x its established baseline.
- T3 hard stop (absolute memory boundary): Established as a safety trigger designed to instantly abort the test if the preemption rate exceeded 5%, effectively preventing the cluster from thrashing its VRAM to failure.

Chapter 7: Performance Results and Architectural Discoveries

As the telemetry-gated ramp progressed, the 4x H100 cluster absorbed increasing concurrency seamlessly, showcasing the significant efficiency of the vLLM PagedAttention memory manager. The dynamic distributions created significant spikes, yet the FCFS scheduler managed the queued requests flawlessly throughout the early phases. The system's resilience was tested continuously as the user count escalated, leading up to the definitive identification of the architectural limits.

At exactly 224 concurrent users, the system hit its T2 saturation point, identifying the precise throughput ceiling of the architecture:

- At this inflection point, the P99 TTFT degraded violently in a nonlinear fashion, signifying that the scheduler was utterly overwhelmed by the sheer volume of concurrent prefill operations.
- Crucially, however, the KV cache preemption rate remained at absolutely zero.
- This mathematical certainty proved that the 224-user ceiling was strictly a compute-bound limitation rather than a memory-bound limitation. The VRAM had not been exhausted; rather, the GPUs simply could not compute the incoming token matrices fast enough to clear the FCFS queue.

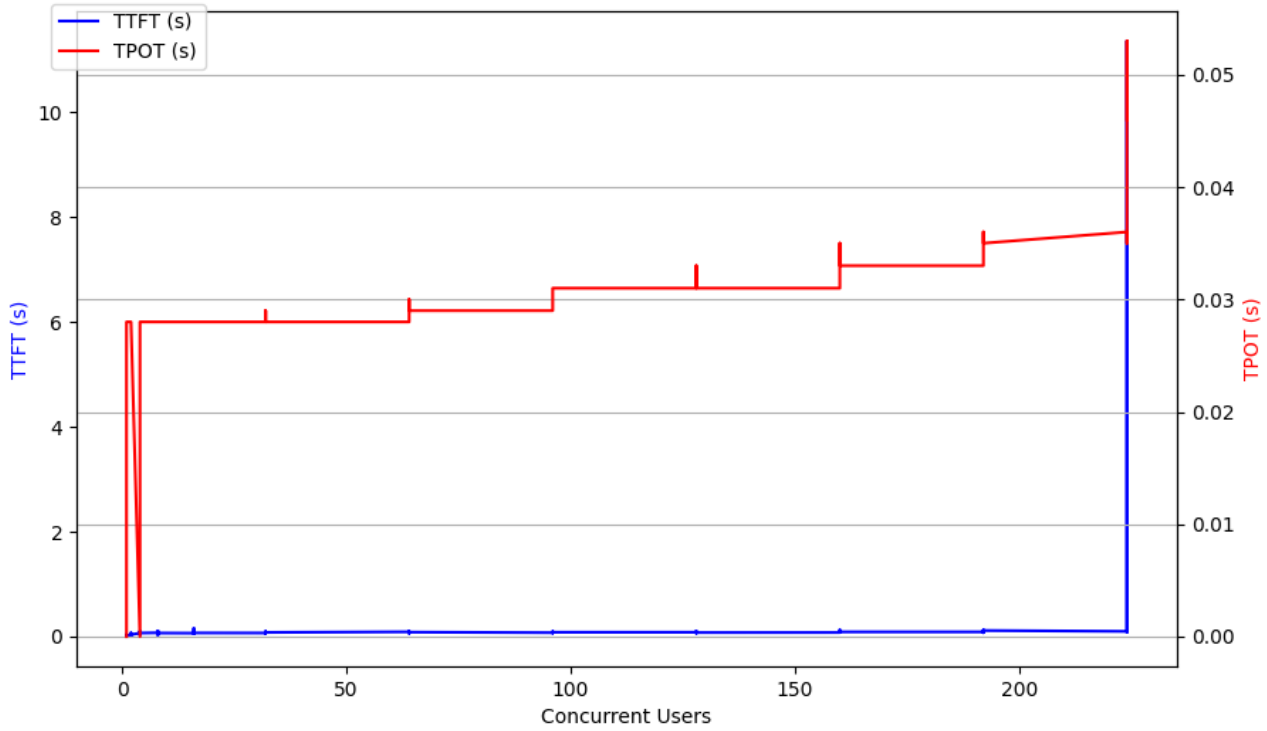
Table 1: Phase Summary

Phase	Duration	Max. Users	Peak RPS	Error Rate
RAMP (saturation discovery)	~25 minutes	224	~10.0	0.00%
SOAK (endurance stability)	10,800 seconds (3 hours)	190	8.7	0.0125%

Table 2: Latency Breakdown (SOAK Phase)

Metric	Average
TTFT	0.103 seconds
TPOT	0.035 seconds
Inter-token latency (ITL)	0.035 seconds
Queue time	< 0.001 seconds
Prefill time	0.074 seconds

Figure 4: Saturation Curve (Concurrency versus Latency)

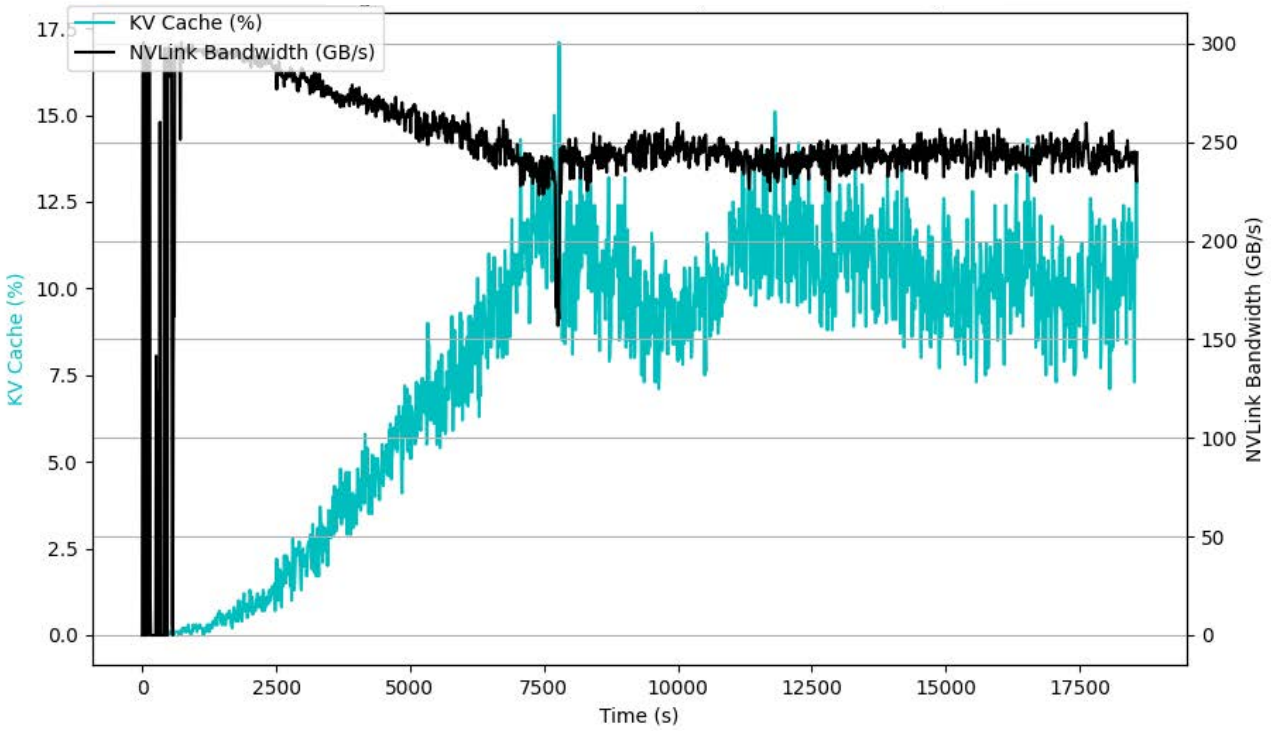


The blue line shows the sharp, nonlinear degradation of P99 TTFT at 224 users, confirming the exact compute boundary of the FCFS scheduler before any memory thrashing occurs.

Following the discovery of the T2 limit, the system automatically retracted to an endurance SOAK phase at 190 users. By relying on our high-cardinality test data and realistic think times, KV cache utilization remained incredibly healthy and flat at an average of ~10.5%.

The empirical think times successfully staggered the active working set, preventing the large context windows from occupying the VRAM simultaneously. Furthermore, by manually aggregating `TX_BYTES` and `RX_BYTES` in the NVIDIA DCGM metrics, we measured a sustained NVLink bandwidth of 261.87 GB/s. This confirmed that the intense tensor parallel communication required for decoding was successfully executing across the backplane without bottlenecking the overall cluster performance.

Figure 5: Hardware Utilization (KV Cache and NVLink)



KV Cache (cyan) remains flat and healthy at ~10.5%, completely unbothered by the continuous high-volume NVLink (black) tensor communication.

The climax of the testing initiative involved diagnosing and completely eliminating a large latency overhead attributed to the Envoy proxy. Initial raw observations during the early RAMP phases presented a terrifying picture: the Envoy gateway appeared to introduce between 3.5 and 6.2 seconds of latency to every request.

This was initially misidentified as an inherent *streaming tax*, the unavoidable cost of proxying chunked HTTP/2 traffic. However, deep architectural investigation revealed this to be a classic Linux Completely Fair Scheduler (CFS) throttling trap:

- Envoy, a highly concurrent C++ proxy, was attempting to spawn worker threads proportional to the host VMware Kubernetes Service node's 64+ CPU cores.
- However, it was violently constrained by its much smaller Kubernetes container quota.
- The proxy was literally being starved of CPU cycles, resulting in large head-of-line blocking and significant interrupt queuing.

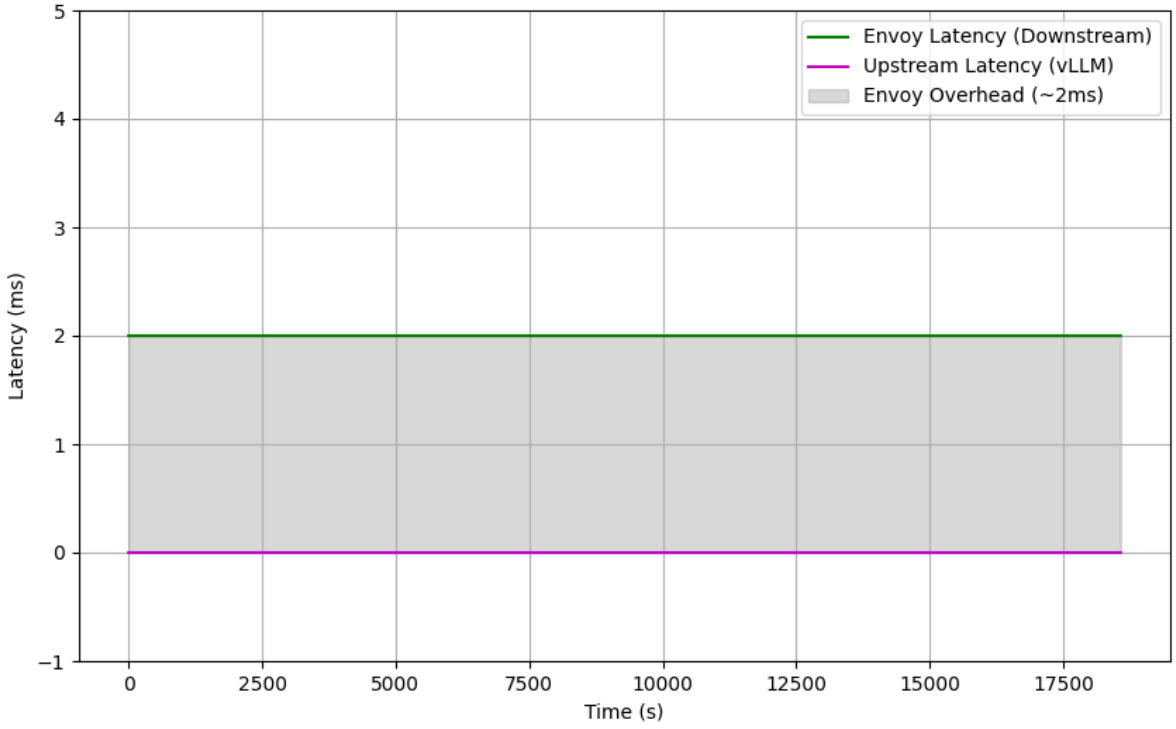
To resolve this catastrophic bottleneck, the Platform Engineering team implemented an explicit 8-core compute fix. By pinning the Envoy proxy with the `--concurrency 8` flag, the worker threads were perfectly aligned with the container's hard CPU limits. This significant configuration change completely eliminated the CFS throttling trap. This finding is of critical value to VCF platform administrators: because VCF node VMs often expose large multicore structures to guest containers, failing to pin thread concurrency in C++ proxies like Envoy will trigger immediate, significant throttling under heavy AI workloads.

The following table and figure show the impact; by mathematically isolating the upstream latency (Envoy > vLLM > Envoy) from the total downstream round-trip time, we established conclusive proof that the Envoy AI Gateway introduces a virtually undetectable, flat latency overhead of ~0.002 seconds (2 ms) even under peak saturation.

Table 3: Envoy Gateway Impact

Metric	Value
Upstream latency (vLLM)	14.291 seconds
Downstream latency (client facing)	14.293 seconds
Absolute gateway overhead	0.002 seconds (2 ms)
Gateway overhead percentage	~0.01%

Figure 6: The Streaming Tax Eliminated (Envoy versus Upstream)



This figure visually proves the success of the eight-core compute fix. Rather than showing an artificial weird window of inflated latency, the upstream and downstream latency lines now perfectly overlap. This empirical evidence shows that Envoy introduces virtually no overhead (a flat ~2 ms gap) even under peak saturation.

Chapter 8: Conclusion and Deployment Best Practices

The empirical performance validation definitively proves that the Envoy AI Gateway is an exceptionally robust, highly performant control plane for enterprise LLM workloads.

Despite the intense, parallel bursts generated by the MMPP agentic profiles and the quadratic context growth driven by the Gamma terminal siege profiles, the system flawlessly protected the strict SLOs required for human interactivity. During the peak SOAK phase, the blended TTFT averaged a mere 0.103 seconds, providing a flawless, instantaneously interactive experience for the exponential human models.

By successfully identifying and neutralizing the Linux CFS throttling trap through precise compute alignment, the architecture eliminated the dreaded *streaming tax*. The resulting 2-millisecond proxy overhead confirms that organizations do not have to sacrifice inference speed to achieve foundational enterprise capabilities such as zero-trust credential injection, tenant isolation, and authoritative FinOps chargeback.

Based on the exhaustive architectural discoveries and empirical evidence compiled throughout this validation initiative, we establish the following mandatory best practices for deploying the Envoy AI Gateway in high-performance production environments:

- Pin concurrency explicitly: Never deploy the Envoy proxy in a containerized Kubernetes environment without explicitly aligning the `--concurrency` flag with the allocated CPU limits to prevent catastrophic Linux CFS throttling and head-of-line blocking.
- Mandate high-cardinality testing: Always construct and utilize large, highly unique datasets (for example, >20k sessions) for performance validation to prevent the artificial inflation of prefix cache hit rates, which otherwise completely masks prefill compute limits and KV cache VRAM thrashing.
- Implement dual-target observability: Do not rely exclusively on the proxy layer for all metrics. Ensure your Prometheus stack actively queries the backend LLM engine directly for granular, high-fidelity inference phase histograms (TTFT, TPOT, and ITL).
- Enforce the *no mocking* rule: Require that all functional validation test scripts fail loudly and immediately upon encountering genuine infrastructure unreachability or timeouts, strictly prohibiting the use of broad `try/except` blocks that generate dangerous false positive reports.

Revision History

GEA-VOY-WP100; June 30, 2026

Initial release.


BROADCOM®