Power System Protection and Control
ISSN:1674-3415

PERFORMANCE ENGINEERING FOR LLM INTEGRATED APPLICATIONS.

Gunjan Shegade

122 Corbin ave, apt 302, Jersey City, NJ 07306
gunjanshegade@gmail.com

Received:19 Oct 2025 **Revised**:21 Nov 2025 **Accepted**: 20 Dec 2025

***ABSTRACT*:**

The integration of large language model (LLM) application programming interfaces into the critical request path of production software introduces a class of performance-engineering problems that classical deterministic system design does not anticipate. Unlike conventional remote procedure calls, an LLM completion call is stochastic in output, autoregressive in structure, and highly variable in latency, since response time scales with generated token count, provider queueing conditions, and model routing decisions rather than fixed computation. This paper characterizes the sources of non-determinism and latency variability in LLM-integrated systems and proposes a structured performance-engineering and testing framework organized around four pillars: token-streaming instrumentation, adaptive timeout strategy, retry-storm-resistant backoff design, and fault-injection-based resilience testing. The framework treats time-to-first-token (TTFT) and inter-token latency (ITL) as first-class service-level indicators, applies deadline propagation across multi-hop agentic call chains, and formalizes retry budgets, jittered exponential backoff, and circuit breakers as the minimum viable resilience surface for AI-dependent services. A comparative synthesis of resilience patterns and a latency-budget taxonomy for four common LLM interaction classes are presented. The discussion argues that performance testing for LLM-integrated applications must shift from throughput-oriented load testing toward probabilistic, chaos-driven validation that treats variable latency and partial failure as expected operating conditions rather than exceptional ones.

***Keywords**: Performance Engineering, Large Language Models, Non-Determinism, Latency Testing, Token Streaming, Timeout Strategy, Retry Storms, Circuit Breaker, Chaos Engineering, Resilience Patterns.*

INTRODUCTION

Modern software systems increasingly place a call to a hosted large language model directly inside the synchronous or near-synchronous request path: a chat assistant, a code-completion tool, a document-summarization pipeline, or a multi-step autonomous agent all depend on at least one network round trip to an inference provider before the system can return a useful result to the user. This architectural choice imports a set of behaviors that traditional backend services rarely exhibit simultaneously. First, the output of the call is not deterministic even when temperature and sampling parameters are held constant, because floating-point non-associativity, GPU kernel scheduling, mixture-of-experts routing, and continuous-batching optimizations at the serving layer all introduce run-to-run variation.

[1] Gate-all-around transistor, 2D materials, sub-2nm technology, AI accelerator, nanosheet, semiconductor scaling. [1] The relentless pace of AI workloads has pushed semiconductor manufacturers to the physicallimits of silicon scaling. Gate-all-around (GAA) transistor architectures have replaced FinFETsat the leading nodes because they offer superior electrostatic control at ultra-short channel lengths, but as the industry moves toward the 2 nm node and beyond, even nanosheet GAAdevices built from silicon face fundamental limits.

[2]Neuromorphic computing, spiking neural networks, task scheduling, green cloud computing, energy efficiency, sustainable data centers. [2] Cloud data Centers now consume roughly 1 to 2 percent of global electricity, and the pressureto reduce that footprint has moved from a corporate sustainability concern to a hard operational constraint. Task scheduling sits at the heart of the problem because every placement decision shapes how much energy the underlying servers, cooling systems, and networking fabric will draw.

DOI: 10.46121/pspc.53.4.42

Second, and central to the concerns of this paper, the latency of the call is itself a random variable with a heavy right tail: because generation is autoregressive, total response time is approximately proportional to the number of output tokens, and it is further inflated by provider-side queueing, rate-limit backpressure, and network conditions that are outside the caller's control. A request that is fast in the 50th percentile can be an order of magnitude slower in the 99th percentile, and that tail is precisely where user experience, cost, and cascading system failure concentrate. Third, because the call is remote and rate-limited, transient errors — 429 (rate limited), 500/502/503 (upstream failure), and stream-level truncations — are routine rather than exceptional, which means that retry logic, if implemented naively, can itself become the dominant cause of an outage through a phenomenon commonly termed a retry storm.

Traditional performance engineering assumes that a service call has a bounded, largely predictable latency distribution and a binary success/failure outcome that can be tested with deterministic fixtures. LLM-integrated systems violate both assumptions simultaneously: latency is heavy-tailed and workload-dependent, and correctness itself is graded rather than binary. This paper argues that these two properties — non-deterministic output and variable, heavy-tailed latency — must be engineered for jointly rather than treated as separate concerns, because the mechanisms that mitigate one (for example, retrying a call whose output failed a quality check) directly affect the other (retrying adds load and tail latency, and can trigger a retry storm against an already-degraded provider).

The remainder of this paper is organized as follows. Section II reviews related work on non-determinism in LLM systems, resilience patterns in distributed systems, and streaming latency measurement. Section III characterizes the specific sources of non-determinism and latency variability that a performance engineer must account for. Section IV presents the proposed testing and engineering framework, covering token-streaming instrumentation, timeout strategy, retry and backoff design, circuit breakers and bulkheads, and chaos-based fault injection. Section V proposes an evaluation framework and metric taxonomy, including a latency-budget table across interaction classes and a comparative table of resilience patterns. Section VI discusses the practical trade-offs and limitations of the framework, and Section VII concludes with directions for future work.

RELATED WORK

Non-determinism in LLM systems has been documented from several angles. Guild.ai's engineering glossary notes that no major inference provider currently promises fully deterministic outputs, and recommends that engineering teams design for tolerance of minor output variation rather than bit-perfect reproducibility, borrowing chaos-engineering practice — deliberately injecting failures and latency perturbations — to probe resilience before it is needed in production [1]. A practical evaluation guide for LLM-reliant systems similarly traces non-determinism to stochastic computation, chained model calls, and grounding-data inconsistencies, and recommends repeated-sampling strategies such as self-consistency to estimate and average out system noise, while cautioning that this approach multiplies the number of calls and therefore directly worsens cost and latency, illustrating the coupling between correctness engineering and performance engineering that motivates this paper [2].

A closer empirical investigation into so-called "deterministic" LLM settings found that even after eliminating GPU-parallelism randomness through fixed seeds, serving-side engineering optimizations such as continuous batching, chunked prefill, and prefix caching can reintroduce non-deterministic behavior, and that this instability propagates into unit-test failures, malformed output formats, and downstream parser errors [3]. This finding is significant for performance engineers because it shows that non-determinism is not solely a property of the model but also of the performance optimizations applied at the serving layer — the very techniques used to reduce latency can increase output variance.

On the testing-methodology side, recent work proposes automated self-testing as a continuous quality gate for LLM applications, arguing that traditional deterministic assertions used in unit and end-to-end testing are poorly suited to probabilistic systems and must be replaced with tolerance-based regression checks [4]. A related empirical study of practitioners testing LLM-powered software confirms that identical prompts can yield materially different outputs, that behavioral drift occurs silently when providers update models without any change to client code, and that teams frequently fall back on manual review because automated black-box and prompt-probing techniques remain limited in scale [6]. A broader reproducibility study of commercial LLM performance in software-engineering research further notes that both epistemic and aleatoric uncertainty vary

DOI: 10.46121/pspc.53.4.42

across models and tasks, making a single general-purpose function for estimating non-determinism unrealistic, and that time-dependent model versioning compounds the reproducibility problem for any longitudinal performance study [7].

On the latency side specifically, LatencyPrism proposes online, non-intrusive latency "sculpting" for large-scale LLM inference serving, motivated by the observation that mainstream providers now embed percentile latency directly into service-level agreements — for instance, aggregating 50th-percentile request latency over rolling intervals — and that conflating queueing delay with actual model execution time obscures the true bottleneck when a service degrades [5]. This distinction between extrinsic scheduling overhead and intrinsic execution latency is directly relevant to the timeout-strategy design proposed in Section IV, because a timeout policy that does not distinguish queueing delay from generation time will either fire prematurely during load spikes or fail to fire when the model itself has stalled.

A large body of resilience-engineering literature outside the LLM context establishes the retry, backoff, circuit-breaker, and bulkhead patterns that this paper adapts to the LLM setting. Guidance on retries in distributed systems consistently identifies the retry storm — where many clients retry simultaneously against an already-struggling dependency — as one of the most common self-inflicted outage mechanisms, and recommends exponential backoff combined with randomized jitter to desynchronize retries, together with hard caps on both delay and attempt count [9]–[14]. A systematic review of recovery patterns across microservice architectures similarly finds that circuit breakers, popularized by Netflix's Hystrix framework, protect dependent services from cascading failure by tripping after a threshold of errors and probing recovery through a half-open state, and that these patterns are frequently combined with bulkhead isolation to prevent one failing dependency from exhausting shared resource pools [21]–[24]. Chaos engineering — the deliberate injection of failure and latency into a running system to validate these patterns before an incident occurs — has likewise been studied as a systematic testing discipline for microservice resilience, with recent work proposing lightweight, model-driven approaches to generate realistic fault scenarios for graph-structured service topologies [22].

Finally, guidance on token streaming and time-to-first-token (TTFT) measurement has emerged primarily from practitioner and vendor sources rather than peer-reviewed venues, reflecting how recent this operational concern is. Server-sent events (SSE) are widely adopted as the transport mechanism for incremental token delivery because they map naturally onto a single long-lived HTTP connection and require no additional client infrastructure beyond the standard EventSource API [16]. Benchmarking guidance distinguishes TTFT — the latency until the first generated token is observed — from inter-token latency (ITL, also called time-per-output-token), and recommends tracking both as separate first-class metrics rather than collapsing them into a single end-to-end duration, since a user's perceived responsiveness is dominated by TTFT even when total generation time is unchanged [15], [17]–[19]. This body of work collectively motivates the framework proposed in the next two sections.

CHARACTERIZING NON-DETERMINISM AND LATENCY VARIABILITY

A. Output Non-Determinism

Even at a sampling temperature of zero, repeated calls to the same hosted model with the same prompt are not guaranteed to return byte-identical output. Contributing factors include floating-point non-associativity across GPUs operating in parallel, non-deterministic kernel-launch ordering, mixture-of-experts routing decisions that can depend on co-scheduled requests in the same batch, and serving-layer optimizations such as continuous batching and prefix caching that were introduced purely for throughput and latency benefit but have the side effect of altering numerical execution order [3]. From a performance-engineering standpoint, the important consequence is not the non-determinism itself but its interaction with correctness-driven retries: a system that retries a call whenever the returned output fails a downstream validation check will retry more often precisely when the model is behaving unusually, which is also often when the serving infrastructure is under load — coupling a correctness problem to a load problem at the worst possible time.

B. Latency Variability and the Heavy Tail

Total response latency for a streamed LLM call is well approximated as the sum of time-to-first-token and the product of output token count and inter-token latency. Because output length is itself a random variable determined by the model's own generation decisions, total latency inherits a heavy right tail even when TTFT and ITL are individually well-behaved. Practitioner benchmarking commonly targets a TTFT under roughly 600 milliseconds

DOI: 10.46121/pspc.53.4.42

for interactive use, since responses that begin streaming within about one second are perceived as responsive while a wait beyond two seconds is noticed by users regardless of how quickly the remainder of the response subsequently streams [19]. Because the perceived and measured components of latency diverge this sharply, a performance test suite that reports only mean or median end-to-end latency is measuring the wrong thing for an interactive LLM-integrated feature; it must separately characterize the TTFT distribution, the ITL distribution, and the tail (p95/p99) of both [17], [18].

C. Multi-Hop Amplification in Agentic Pipelines

The variability problem compounds when a single user-facing request triggers a chain of LLM calls, as in retrieval-augmented generation or autonomous agent pipelines that plan, call a tool, and then synthesize a final answer. Each hop in such a chain has its own TTFT and ITL distribution, and because the hops are typically sequential, the end-to-end latency distribution is the convolution of several heavy-tailed distributions rather than a single one. Observability guidance for agentic systems emphasizes separating gateway timing, model timing, tool-call timing, and client delivery time precisely because a single slow hop deep in the chain can dominate the entire user-perceived latency even if every other hop is fast, and because streaming the wrong phase of a multi-step pipeline can make a system feel slower than an equivalent single LLM call that streams immediately [15].

D. Provider-Side Failure Modes

Beyond latency variance, LLM API calls fail in ways that are routine rather than exceptional: rate-limit rejections (HTTP 429) during traffic spikes, transient upstream errors (5xx) during provider-side incidents or model rollouts, and stream-level failures such as a connection dropping mid-generation after several hundred tokens have already been received and billed. Any of these failure modes can be encountered by any given request with non-trivial probability at production scale, which means that the failure-handling logic — not the happy path — determines the tail behavior of the overall system. This is the central justification for treating retry, timeout, and circuit-breaking policy as core performance-engineering artifacts rather than incidental error handling.

PROPOSED PERFORMANCE-ENGINEERING AND TESTING FRAMEWORK

We propose a framework organized around four complementary engineering disciplines, each of which must be instrumented, tested, and validated independently and then re-validated together under combined fault injection, since the interactions between them (for example, retries interacting with timeouts, or circuit breakers interacting with streaming) are frequently where production incidents originate.

A. Token-Streaming Instrumentation

Streaming delivers generated tokens to the client incrementally over a single long-lived connection, typically using server-sent events over chunked HTTP transfer encoding, rather than waiting for the complete response before returning anything [16], [22]. The performance-engineering justification for streaming is not that it reduces total generation time — it does not — but that it moves the first user-visible content far earlier in the request lifecycle, which dominates perceived responsiveness. A response that begins rendering at 400 milliseconds and continues streaming for several seconds is judged by users to be faster than a response that returns everything at once after an equivalent total duration, because attention and reading time overlap with the remaining generation [19], [20].

For testing and monitoring purposes, we recommend instrumenting three timestamps at the gateway boundary for every streamed call: request-dispatch time, first-chunk-received time, and final-chunk-received time. From these, TTFT is computed as the interval between dispatch and first chunk, and ITL (equivalently, time-per-output-token) is computed as the interval between first and final chunk divided by the number of output tokens, explicitly excluding TTFT from the per-token average so that a slow start does not distort the measured generation rate [17]. A common instrumentation defect is re-buffering: a server that consumes a provider's stream internally and only forwards a complete response to its own client discards the entire latency benefit of streaming while still paying its cost in implementation complexity, so end-to-end TTFT should always be measured from the outermost client boundary, not from the internal provider call [19].

B. Adaptive Timeout Strategy

A single fixed timeout applied uniformly to all LLM calls is inappropriate given the heavy-tailed, workload-dependent nature of generation latency: a timeout tight enough to bound interactive chat latency will spuriously cancel long-form summarization calls, while a timeout loose enough to accommodate long-form generation will

DOI: 10.46121/pspc.53.4.42

let an interactive feature hang far past the point of user abandonment. We propose a timeout strategy with three components. First, a per-class latency budget is set according to interaction type (see Table I in Section V), reflecting that autocomplete, conversational chat, agentic tool steps, and batch summarization have fundamentally different tolerance for delay. Second, timeouts should be split into a TTFT sub-timeout and an overall stream-deadline: if no chunk arrives within the TTFT sub-timeout, the call is almost certainly experiencing a provider-side stall or network partition rather than long generation, and can be cancelled and retried immediately without waiting for the full deadline to elapse; if chunks are arriving steadily but the overall deadline is approached, the client should truncate gracefully rather than abort, since partial output is frequently still useful. Third, in multi-hop agentic pipelines, the remaining time budget should be propagated explicitly from the outer request deadline into each inner call — deadline propagation — so that a tool-calling step deep in a chain does not consume a timeout budget appropriate for a standalone call while ignoring how much of the user's overall patience has already elapsed [5], [15].

C. Retry Policy Without Retry Storms

Because transient 429 and 5xx responses from LLM providers are routine, some retry logic is necessary; the risk is that naive retry logic — immediate, unbounded, unjittered — is one of the most common ways that distributed systems accidentally attack themselves during a partial outage, since many clients failing and retrying at the same moment can convert 1,000 requests per second of legitimate load into an order of magnitude more retry traffic against an already-struggling dependency, a pattern termed a retry storm [12], [13]. The standard mitigation, adapted here for LLM calls, has four elements. First, only errors classified as transient (429, 502, 503, 504, and stream-level disconnects) should be retried; errors that indicate a permanent client-side problem (400, 401, content-policy rejection) should fail fast [14]. Second, retry delay should follow exponential backoff with full jitter, computed as a random draw between zero and the minimum of a capped maximum delay and the base delay doubled per attempt, which spreads retries in time rather than clustering them at fixed intervals [9]–[11]. Third, both the maximum delay (typically bounded to tens of seconds) and the maximum attempt count (typically three to five attempts) must be capped, since unbounded retries eventually exhaust client-side resources even if they never overload the provider [14]. Fourth, and specific to LLM calls, retries should be governed by a retry budget that limits total retry volume to a small fraction — commonly cited as ten to twenty percent — of live traffic; once the budget is exhausted, the system should fail fast rather than retry, protecting the downstream provider from exactly the amplification a retry storm produces [13].

A further LLM-specific nuance is idempotency and partial-output handling. Because a streamed call that fails after several hundred tokens have already been generated and billed cannot simply be re-issued from scratch without financial and latency cost, retry logic for LLM APIs should distinguish between a failure before any tokens were received (safe to retry transparently) and a failure mid-stream after substantial partial output exists (better handled by resuming with the partial output as context, or by surfacing the partial result to the user with an option to continue, rather than by discarding it and retrying blind) [12].

D. Circuit Breakers and Bulkhead Isolation

Retry policy alone does not prevent cascading failure if the upstream provider or an internal LLM-gateway component is degraded for an extended period; in that situation, every retry — however well-jittered — still adds load to a system that needs to recover, not receive more traffic. The circuit-breaker pattern, popularized in distributed systems by Netflix's Hystrix framework, addresses this by monitoring the error rate of calls to a dependency and tripping to an open state once a failure threshold is exceeded, after which calls fail immediately (or fall back to a default behavior) without being attempted at all; after a cooldown period the circuit enters a half-open state that allows a small number of probe requests through to test whether the dependency has recovered before fully closing the circuit again [21], [23], [24]. Applied to LLM-integrated applications, we recommend that circuit breakers be scoped per model/provider endpoint rather than globally, since a single application commonly depends on more than one model (for example, a fast low-cost model for classification and a larger model for generation), and a failure in one should not degrade the other.

Bulkhead isolation complements circuit breaking by partitioning shared resources — connection pools, thread pools, or concurrency-limited request queues — per dependency, so that a single overloaded or slow model endpoint cannot exhaust the resources available to calls destined for a different, healthy endpoint [21]. In an LLM gateway that fans requests out to multiple providers or model tiers, bulkheads are what prevent a latency spike in one tier from starving concurrency for every other tier.

DOI: 10.46121/pspc.53.4.42

E. Fallback and Degradation Strategy

Every timeout, open circuit, and exhausted retry budget must resolve to a defined fallback rather than an unhandled error, because from the end user's perspective a clear, fast degraded response is preferable to an unbounded wait. Practical fallback options for LLM-integrated features include: serving a cached or previously generated response for repeat or near-duplicate queries; downgrading to a smaller, faster, or self-hosted model with lower quality but bounded latency; returning the partial output already streamed before a mid-stream failure, clearly marked as incomplete; and, for non-interactive paths, queueing the request for asynchronous retry and notifying the user rather than blocking. Which fallback is appropriate depends on the interaction class defined in Table I, and should be decided at design time, not discovered during an incident.

F. Chaos and Fault-Injection Testing Methodology

Because the failure and latency modes described above are routine rather than rare, they cannot be validated solely through conventional unit and integration tests that assume a single, fast, successful call. We recommend chaos-engineering practice — the deliberate, controlled injection of failures and perturbations into a system to test its resilience before those conditions occur naturally in production [1], [22] — as the primary validation methodology for the framework above, applied specifically to the LLM call boundary. A minimal fault-injection test matrix should include: (i) artificially delayed TTFT (simulating provider queueing) at multiple magnitudes relative to the configured timeout, to verify that the TTFT sub-timeout fires correctly and does not falsely trigger under normal jitter; (ii) mid-stream connection termination after a partial token count, to verify that partial-output handling and retry-with-context behave as designed rather than discarding partial work; (iii) sustained 429/503 injection at a rate sufficient to open the circuit breaker, to verify that the breaker trips, that fallback behavior activates, and that the retry budget is not exhausted prematurely by the injected errors; (iv) synchronized-failure injection across many simulated concurrent clients, to verify that jittered backoff actually desynchronizes retries rather than producing a retry storm in aggregate; and (v) combined-fault scenarios that inject latency and errors simultaneously across multiple hops of an agentic pipeline, to verify that deadline propagation correctly apportions the remaining time budget rather than allowing an early hop to consume the entire allowance. This test matrix should be run continuously in a staging environment against a simulated or record-and-replay provider, since testing directly against a live production LLM provider's actual failure modes is neither reliable nor ethical as a repeatable test methodology.

EVALUATION FRAMEWORK AND METRICS

Consistent with the observability guidance reviewed in Section II, we recommend treating TTFT and ITL as separate first-class service-level indicators rather than collapsing them into a single end-to-end latency figure, and reporting both at the median and at the 95th and 99th percentiles, since tail behavior — not the median — is what determines whether timeout and retry policy is correctly calibrated [15], [17]–[19]. Table I proposes an illustrative latency-budget taxonomy across four common LLM interaction classes, intended as a starting template for teams to adapt to their own product's tolerance rather than as a universal standard.

Interaction Class	Target TTFT	Acceptable ITL	Timeout Budget	Fallback on Breach
Conversational chat	< 600 ms	20–60 ms/token	8–15 s	Cached reply / degraded model
Autocomplete / inline suggestion	< 300 ms	10–30 ms/token	1–3 s	Suppress suggestion
Agentic tool-calling step	< 1.5 s	n/a (non-streamed)	20–45 s	Re-plan with smaller model
Batch / offline summarization	< 5 s	50–150 ms/token	60–180 s	Queue for retry

TABLE I. Illustrative Latency Budgets and Fallback Policy by LLM Interaction Class

Table II synthesizes the resilience patterns discussed in Section IV, mapping each pattern to the specific failure mode it addresses, its underlying mechanism, and the risk introduced if it is misconfigured — a risk that is easy to overlook because a resilience pattern that fails silently (for example, a circuit breaker whose threshold is set too high to ever trip) provides no warning until the failure it was meant to prevent has already occurred.

DOI: 10.46121/pspc.53.4.42

Resilience Pattern	Primary Failure Mode Addressed	Mechanism	Risk if Misconfigured
Exponential backoff	Transient 429 / 5xx from provider	Delay doubles per attempt up to a cap	Long tail latency, thundering herd on recovery
Full jitter	Synchronized retries across clients	Random delay drawn from [0, backoff]	Insufficient spread if entropy is shared
Retry budget	Retry amplification / storms	Cap retries to 10–20% of live traffic	Silent request drops if budget starved
Circuit breaker	Cascading failure to upstream services	Trip on error-rate threshold; half-open probe	Flapping if thresholds too sensitive
Bulkhead isolation	One tenant/model exhausting shared pool	Separate connection/thread pools per dependency	Under-utilization if pools too small
Deadline propagation	Wasted work past client patience	Pass remaining time budget through call chain	Premature cancellation of near-complete work

TABLE II. Comparative Summary of Resilience Patterns for LLM-Integrated Call Paths

Beyond the two tables above, we recommend three aggregate metrics for ongoing performance-engineering review. First, the retry amplification ratio — the ratio of total calls actually issued (including retries) to the number of distinct user-initiated requests — should be tracked continuously; a ratio approaching or exceeding the configured retry budget is an early warning of an incipient retry storm before it becomes visible as a user-facing outage [12], [13]. Second, the circuit-breaker duty cycle — the fraction of time each provider-scoped breaker spends open versus closed — indicates how frequently a dependency is unhealthy and whether the trip threshold is well calibrated. Third, the streamed-completion rate — the fraction of streamed responses that complete without a mid-stream failure — directly measures how often partial-output handling logic is exercised in production and should be cross-referenced against the fallback strategies defined in Section IV-E.

DISCUSSION

The framework proposed in this paper reflects a broader shift that LLM integration forces onto performance engineering: variable latency and partial failure must be treated as expected, routinely-occurring operating conditions to be engineered around, rather than exceptional cases to be handled defensively as an afterthought. This shift has direct organizational consequences. Load testing for LLM-integrated features cannot rely solely on throughput benchmarks against a stable, fast backend, because the dominant source of tail latency and failure is not the caller's own infrastructure but a third-party dependency whose internal queueing and routing behavior is opaque to the caller [5]. This motivates record-and-replay or simulated-provider testing, described in Section IV-F, as a necessary complement to live integration testing.

A second consequence is that the resilience patterns discussed in Section IV interact with each other in ways that must be tested jointly, not merely unit-tested in isolation. A retry policy that is correctly jittered in isolation can still contribute to a retry storm if the circuit breaker it should be coordinating with has a threshold set too high to trip before the retry budget is exhausted [13], [14]. Similarly, a timeout policy that is well calibrated for a single call can behave incorrectly inside a multi-hop agentic pipeline if deadline propagation is not implemented, since each hop will independently apply its full local timeout budget rather than the fraction of the user's remaining patience actually available [15]. This argues for the combined-fault test scenarios recommended in Section IV-F as the primary validation gate, rather than treating streaming, timeout, retry, and circuit-breaking configuration as four independently-reviewed settings.

A limitation of the framework as presented is that it does not resolve the tension, noted in Section II, between correctness-oriented mitigations for non-determinism (such as self-consistency sampling, which issues multiple calls per request to reduce output variance) and performance-oriented mitigations for latency and load (such as retry budgets, which explicitly cap additional call volume) [2]. In practice, teams must decide, per interaction

DOI: 10.46121/pspc.53.4.42

class, whether an unreliable output is better addressed by consuming additional latency and cost for a second sample, or by accepting the variance and validating downstream. This trade-off is inherently product-specific and is not resolved by the infrastructure-level patterns in Section IV alone; it requires the latency-budget and quality-tolerance decisions summarized in Table I to be made deliberately at the product level, not defaulted implicitly by whichever engineer configures the retry client.

Finally, the framework assumes that the calling application has sufficient control over its request path to implement deadline propagation, per-provider circuit breakers, and streaming instrumentation at the gateway boundary. Applications built directly against a provider's client SDK without an intermediating gateway layer will find several of the recommendations in Section IV harder to retrofit, which argues for introducing a thin LLM-gateway abstraction early in a system's lifecycle specifically to centralize this instrumentation, rather than duplicating it inconsistently across every calling service.

CONCLUSION AND FUTURE WORK

This paper has characterized the joint problem of output non-determinism and heavy-tailed, variable latency that arises when large language model API calls are placed in the critical path of production applications, and has proposed a four-part performance-engineering framework covering token-streaming instrumentation, adaptive timeout strategy with deadline propagation, retry-storm-resistant backoff and retry-budget design, and circuit-breaker/bulkhead-based fault isolation, validated through chaos-style fault injection rather than conventional deterministic testing. A latency-budget taxonomy across four interaction classes and a comparative summary of resilience patterns were presented as practical starting artifacts for teams adopting this framework. The central argument is that these four disciplines cannot be engineered or tested independently, because their interactions — for example, retries against an untripped circuit breaker, or local timeouts inside a multi-hop pipeline without deadline propagation — are frequently the actual source of production incidents, even when each component appears correctly configured in isolation.

Future work should pursue empirical validation of the illustrative latency budgets in Table I against production telemetry across a range of LLM-integrated application types, development of open tooling for record-and-replay simulation of provider-side failure and latency distributions specifically for use in continuous chaos testing, and closer integration between the correctness-oriented evaluation methodologies discussed in Section II (such as self-consistency sampling) and the latency- and cost-aware retry-budget mechanisms proposed in Section IV, so that the trade-off between output reliability and system performance can be tuned quantitatively rather than left to ad hoc per-team judgment.

REFERENCES

1. **Pawan Kumar Singh,** Scaling Gate-All-Around (GAA) Transistor Architectures for Sub-2nm AI Accelerators Using Atomically Thin 2D Materials, Vol. 33 No. 08 (2024): JOCAAA, Journal Name: Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5713
2. **Pawan Kumar Singh,** Energy-Efficient Task Scheduling in Green Cloud Computing Using Neuromorphic Spiking Neural Networks, Vol. 33 No. 08 (2024): JOCAAA, Journal Name: Journal of Computational Analysis and Applications, Journal's ISSN: 1521-1398 (Paper),1572-9206 (Online), https://eudoxuspress.com/index.php/pub/article/view/5712
3. Bandari, V. V., & Lekkala, H. B. (2024). AI-based operator behavior monitoring and cost optimization using digital traceability in manufacturing systems. *Power System Protection and Control*, 52(1), 38–45. https://pspac.info/index.php/dlbh/article/view/342
4. Mayank Atreya, Navin Chhibber, Harvendra Singh, Explainable Machine Learning For Dynamic Pricing In Fast-Changing Retail Environments, 2022/4/9, Journal ,Available at SSRN 6011354, https://scholar.google.com/citations?view_op=view_citation&hl=en&user=fyViF1UAAAAJ&citation_for_view=fyViF1UAAAAJ:LkGwnXOMwfcC.

DOI: 10.46121/pspc.53.4.42

5. Venumadhav Vavilala, Shankar Balla (2024, May). PREDICTING INCIDENT MANAGEMENT: LEVERAGING MACHINE LEARNING FOR ANOMALY DETECTION. Power System Protection and Control Scopus Q1 Journal. PSPC. https://pspac.info/index.php/dlbh/article/view/270
6. Godavari Modalavalasa, LARGE LANGUAGE MODELS FOR INTELLIGENT DATA ENGINEERING: AUTOMATING SCHEMA DESIGN, LINEAGE, AND QUALITY CONTROL, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/183 , DOI: https://doi.org/10.46121/pspc.50.2.4
7. Godavari Modalavalasa, FEDERATED LEARNING FOR ENTERPRISE CLOUD DATA ENGINEERING: ARCHITECTURE, SECURITY, AND GOVERNANCE CHALLENGES, Vol. 51 No. 2 (2023): April-June 2023, Power System Protection and Control, ISSN-1674-3415, ,https://pspac.info/index.php/dlbh/article/view/184, DOI: https://doi.org/10.46121/pspc.51.2.5
8. Godavari Modalavalasa, AI-DRIVEN DATA GOVERNANCE: INTELLIGENT METADATA, LINEAGE, AND COMPLIANCE AUTOMATION IN CLOUD DATA PLATFORMS, Archives / Vol. 52 No. 1 (2024): January-March 2024 /, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/182, DOI: https://doi.org/10.46121/pspc.52.1.3
9. Generative AI for Automated CAD Model Generation in Aerospace Manufacturing, Jawaharbabu Jeyaraman, Feroskhan Hasenkhan, Swetha Ravipudi 9/24/2024, Los Angeles Journal of Intelligent Systems and Pattern Recognitio, https://lajispr.org/index.php/publication/article/view/19
10. Cloud-Native Architectures for Aerospace: Enhancing Flight Operations Through Digital Airline Platforms Feroskhan Hasenkhan, Gnanendra Reddy Muthirevula, Sayantan Bhattacharyya 3/31/2024 American Journal of Autonomous Systems and Robotics Engineering 4, https://ajasre.org/index.php/publication/article/view/25
11. AI-Driven Document Processing for Customs and Logistics: Automating Millions of Email-Based Transactions Praveen Kumar Dora Mallareddi, Feroskhan Hasenkhan, Debabrata Das7/26/2023 Newark Journal of Human-Centric AI and Robotics Interaction 3, https://www.njhcair.org/index.php/publication/article/view/13
12. Manikantha Varaprasad Inakollu, (2024, May), FINOPS-Driven Cloud optimization models for enterprise applications, Vol. 52 No. 2 (2024): April-June 2024, 99-110, Power System Protection and Control, ISSN-1674-3415, URL: https://pspac.info/index.php/dlbh/article/view/283 DOI: https://doi.org/10.46121/pspc.52.2.10
13. Naresh Lokiny. (2022). Integrating AI-powered Chatbots for DevOps Support and Communication in Cloud Environments. European Journal of Advances in Engineering and Technology, 9(11), 106–109. https://doi.org/10.5281/zenodo.13325989
14. Naresh Lokiny, (2021), "Disaster Recovery and Business Continuity Planning in DevOps Cloud with AI", International Journal of Science and Research (IJSR), 10(3), 2024-2027. https://dx.doi.org/10.21275/SR24724151733, https://www.ijsr.net/getabstract.php?paperid=SR24724151733

DOI: 10.46121/pspc.53.4.42

15. Naresh Lokiny, & Ranganath Nandanampati. (2020). DevSecOps: Integrating Security into DevOps with AI in Cloud. Journal of Scientific and Engineering Research, 7(10), 239–242. https://doi.org/10.5281/zenodo.13348695
16. Naresh Lokiny, & Pradip Reddy. (2021). Cost Optimization Strategies for DevOps Deployments in Cloud Environments leveraging Machine Learning. European Journal of Advances in Engineering and Technology, 8(3), 69–72. https://doi.org/10.5281/zenodo.13325845
17. Jayanth Para, 6G Internet Technology Cyber Threat Notification & Alert System, Vol. 52 No. 4 (2024): October-December 2024, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/170 , DOI: https://doi.org/10.46121/pspc.52.4.9
18. Jayanth Para, AI Based Cloud Computation Observational Method & Process, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/171 , DOI: https://doi.org/10.46121/pspc.51.4.3
19. **Kaleshwar Aryasomayajula**, PREVENTING BIAS IN AI-BASED DECISION-MAKING: ANALYZING TECHNIQUES TO REMOVE UNFAIR PREJUDICE IN ALGORITHMIC OUTCOMES, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/294
20. Kaleshwar Aryasomayajula, MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/364
21. **Kaleshwar Aryasomayajula**, Micro services: "A Comparative Analysis of Monolithic vs. Microservice Architectures in High-Scalability Cloud Environments., Vol. 51 No. 2 (2023): April-June 2023 , Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/374
22. **Vikas Suresh Bagora**, KERNEL-LEVEL PERFORMANCE OPTIMIZATION FOR CARRIER-GRADE BROADBAND AND HIGH-SPEED NETWORKING SYSTEMS, Vol. 47 No. 1 (2019), Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/359
23. **Vikas Suresh Bagora,** SCALABLE 128G FIBRE CHANNEL AND ETHERNET CONVERGENCE FOR NEXT-GENERATION DATA CENTER NETWORKS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/362
24. **Vikas Suresh Bagora**, SECURE EMBEDDED NETWORKING ARCHITECTURES USING TRUSTED EXECUTION ENVIRONMENTS IN BROADBAND GATEWAYS, Vol. 52 No. 2 (2024): April-June 2024, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/363
25. Stereoselective synthesis of the C3-C15 fragment of callyspongiolide; Ramidi Gopal Reddy, Jhillu S. Yadav and Debendra K. Mohapatra. (Tetrahedron Letters. 2018, 59, 3579-3582). https://doi.org/10.1016/j.tetlet.2018.08.041, https://www.sciencedirect.com/science/article/pii/S0040403918310426
26. Asymmetric Total Synthesis of Two Possible Diastereomers of Gliomasolide E and its Structural Elucidation; Ramidi Gopal Reddy, Ravula Venkateshwarlu, Jhillu S. Yadav and

Debendra K. Mohapatra. (J. Org. Chem. 2017, 82(2), 1053-1063). https://doi.org/10.1021/acs.joc.6b02611 https://pubs.acs.org/doi/abs/10.1021/acs.joc.6b02611

27. Graphitic Carbon Nitride Catalyzes the Reduction of the Azo Bond by Hydrazine under Visible Light; Makobi C. Okolie, Glory G. Ollordaa, Gopal R. Ramidi, Xin Yan, Yufeng Quan, Qingsheng Wang and Yingchun Li. (Nanomaterials 2024, 14(17), 1402), https://doi.org/10.3390/nano14171402, https://www.mdpi.com/2079-4991/14/17/1402
28. **Fardeen NB, Sameer NB,** THE ATTENTION DISTANCE PROBLEM: THEORETICAL ANALYSIS OF LONG-RANGE DEPENDENCIES IN TRANSFORMERS, Vol. 52 No. 2 (2024): April-June 2024, Power System Protection and Control, ISSN-1674-3415, **https://pspac.info/index.php/dlbh/article/view/313**
29. **Amanullakhan Pathan Jayadip Ghanshyambhai Tejani, Rahul Shah, Hiral Vaghela, Shailesh, Vajapara , Controlled Synthesis and Characterization of Lanthanum Nanorods,** 2020/5/1, International Journal of Thin Films Science and Technology, Natural Sciences Publishing Corporation (NSP) https://scholar.google.com/citations?view_op=view_citation&hl=en&user=efbNMkwAAAAJ&citation_for_view=efbNMkwAAAAJ:M3NEmzRMIkIC
30. Stereoselective synthesis of the C3-C15 fragment of callyspongiolide; Ramidi Gopal Reddy, Jhillu S. Yadav and Debendra K. Mohapatra. (Tetrahedron Letters. 2018, 59, 3579-3582). https://doi.org/10.1016/j.tetlet.2018.08.041, https://www.sciencedirect.com/science/article/pii/S0040403918310426
31. Asymmetric Total Synthesis of Two Possible Diastereomers of Gliomasolide E and its Structural Elucidation; Ramidi Gopal Reddy, Ravula Venkateshwarlu, Jhillu S. Yadav and Debendra K. Mohapatra. (J. Org. Chem. 2017, 82(2), 1053-1063). https://doi.org/10.1021/acs.joc.6b02611, , https://pubs.acs.org/doi/abs/10.1021/acs.joc.6b02611
32. Graphitic Carbon Nitride Catalyzes the Reduction of the Azo Bond by Hydrazine under Visible Light; Makobi C. Okolie, Glory G. Ollordaa, Gopal R. Ramidi, Xin Yan, Yufeng Quan, Qingsheng Wang and Yingchun Li. (Nanomaterials 2024, 14(17), 1402), https://doi.org/10.3390/nano14171402, https://www.mdpi.com/2079-4991/14/17/1402
33. **Kaleshwar Aryasomayajula**, PREVENTING BIAS IN AI-BASED DECISION-MAKING: ANALYZING TECHNIQUES TO REMOVE UNFAIR PREJUDICE IN ALGORITHMIC OUTCOMES, Vol. 50 No. 2 (2022): April-June 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/294
34. Kaleshwar Aryasomayajula, MODERN DEVELOPMENT PRACTICES: INVESTIGATIONS INTO SERVERLESS COMPUTING, LOW-CODE/NO-CODE PLATFORMS, AND DEVELOPER PRODUCTIVITY IN REMOTE ENVIRONMENTS, Vol. 50 No. 4 (2022): October-December 2022, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/364
35. **Kaleshwar Aryasomayajula**, Micro services: "A Comparative Analysis of Monolithic vs. Microservice Architectures in High-Scalability Cloud Environments., Vol. 51 No. 2 (2023): April-June 2023 , Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/374

DOI: 10.46121/pspc.53.4.42

36. :Controlled Synthesis and Characterization of Lanthanum Nanorods, **Author(s), Jayadeep Tejani, Rahul Shah, Hiral Vaghela, Shailesh Vajapara, Amanullakhan Pathan**, doi:10.18576/ijtfst/090205,
URL: https://www.naturalspublishing.com/Article.asp?ArtcID=20705, May-2020
37. **SUDIPKUMAR GHANVAT , AISHWARYA BADLANI, SHREYA SANDEEP JOSHI,** MARKETING EFFECTIVENESS IN THE POST-COOKIE ERA: A COMPARATIVE ANALYSIS OF FIRST PARTY AND ZERO-PARTY DATA STRATEGIES ON CAMPAIGN PERFORMANCE AND CUSTOMER EQUITY, **VOL. 31 NO. 4 (2023):** JOCAAA ,
HTTPS://WWW.EUDOXUSPRESS.COM/INDEX.PHP/PUB/ARTICLE/VIEW/3940
38. Adegboye A. Olaoluwa1, Udofia M. Kufre,Obot Akanniyenne, Inverted C-Shaped Slots Loaded Exponential Tapered Triple Band Notched Ultra-Wideband (UWB) Antenna, https://doi.org/10.5281/zenodo.18139060, **Published May 31, 2024**
39. **Chander Vijay S Sanbhi**, DATA QUALITY AS A RELIABILITY MULTIPLIER: QUANTIFYING THE IMPACT OF STANDARDIZED WORK PROCESSES ON RELIABILITY MODEL ACCURACY, Vol. 52 No. 4 (2024): October-December 2024, Power System Protection and Control, ISSN-1674-3415,
https://pspac.info/index.php/dlbh/article/view/385
40. **Chander Vijay S Sanbhi,** DEGRADATION-AWARE SPARES OPTIMIZATION WITH WEIBULL/PHM UNCERTAINTY AND LOGISTICS DELAYS, Vol. 52 No. 2 (2024): April-June 2024, Power System Protection and Control, ISSN-1674-3415,
https://pspac.info/index.php/dlbh/article/view/384
41. **Chander Vijay S Sanbhi,** DIGITAL TWIN–RAM CO-SIMULATION FOR MAINTENANCE AND SPARING DECISIONS IN PROCESS FACILITIES, Vol. 52 No. 1 (2024): January-March 2024, Power System Protection and Control, ISSN-1674-3415,https://pspac.info/index.php/dlbh/article/view/383
42. **Chander Vijay S Sanbhi,** BAYESIAN UPDATING OF RAM MODELS FOR DYNAMIC AVAILABILITY FORECASTING UNDER REALISTIC DOWNTIME CONSTRAINTS**, Vol. 51 No. 3 (2023): July**-September 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/381
43. **Chander Vijay S Sanbhi,** HYBRID RELIABILITY MODELLING FOR ROTATING EQUIPMENT: PHYSICS-INFORMED LEARNING WITH SPARSE FAILURE DATA, Vol. 51 No. 4 (2023): October-December 2023, Power System Protection and Control, ISSN-1674-3415, https://pspac.info/index.php/dlbh/article/view/382

DOI: 10.46121/pspc.53.4.42