Performance Validation of AI-LLM based Applications *Cloud Rate Limits, Token Management, and Optimization Strategies*

About the Author

Vivek Sharma is an App Development Manager within Accenture products team with over 21 years of experience in Cloud Technologies, Performance Engineering and Validations and current interest on AI/ML. He is a multi-cloud expert and solves daily issues around cloud and AI for clients and internal teams. His expertise spans prompt engineering, token optimisation, and finops in AI, with particular focus on cost optimization and automation.

1. Executive Summary

Large Language Model (LLM)-powered applications have rapidly moved from experimental prototypes to mission-critical enterprise systems. Unlike traditional software, these applications operate under a unique set of performance constraints imposed not by the application tier alone, but by the cloud providers that host and serve the underlying foundation models. Understanding, measuring, and designing around these constraints is now a fundamental discipline in AI application engineering.

This whitepaper provides a practitioner-focused reference for performance validation of LLM-based applications across the three dominant cloud platforms: Amazon Web Services (AWS Bedrock), Microsoft Azure (Azure OpenAI Service), and Google Cloud (Vertex AI / Gemini API). It covers the mechanics of token-based rate limiting (TPM, RPM, TPD), how soft and hard quotas are structured at the account and regional levels, and how these limits interact with application behaviour under load.

Key findings include:

- All three major cloud providers enforce limits across at least two dimensions (TPM and RPM), with Google additionally enforcing daily request limits (RPD). Each provider's throttling behaviour is subtly different and requires provider-specific tuning.
- AWS Bedrock's token reservation model — which deducts quota at request start based on max_tokens, not actual usage — is the most common source of unexpected throttling in high-concurrency applications.
- Azure OpenAI's RPM-to-TPM ratio is fixed at 6 RPM per 1,000 TPM, meaning throughput optimization requires balancing both axes simultaneously.
- Google Vertex AI's dynamic shared quota (DSQ) model and spend-based tier progression require organizations to plan quota growth as part of their FinOps strategy.
- Prompt optimization, context caching, and output-length control consistently yield 30–70% reductions in token consumption without sacrificing application quality.

2. Introduction

The Rise of LLM-Powered Applications

Since 2023, enterprises across every sector have deployed applications built on large language models for use cases ranging from intelligent search and document processing to autonomous agents and multi-turn conversational systems. These applications share a common characteristic: their performance profile is fundamentally governed by the inference service that serves the underlying model, not by the application code alone.

Unlike traditional APIs where latency is measured in milliseconds and throughput in requests per second, LLM APIs operate on a different axis: the token. A single LLM request may consume thousands of tokens, and the cloud platform allocates capacity based on tokens-per-minute (TPM) rather than raw request count. This means performance testing an LLM application requires a different measurement framework than traditional load testing.

Why Performance Validation Is Different for LLM Applications

Traditional performance testing focuses on response time, throughput, and error rates under simulated concurrent load. For LLM applications, these dimensions remain important but are insufficient on their own. Performance validation must also account for:

- **Token consumption variability**: Output length, and thus token cost, varies with each request. The same prompt can yield dramatically different token counts on different invocations.
- **Quota reservation vs. actual consumption**: Several providers reserve quota at request start based on the max_tokens parameter, not the actual tokens generated.
- **Burst vs. sustained throughput**: LLM platforms evaluate rate limits on rolling windows (typically 1–60 seconds), meaning burst traffic patterns can trigger throttling even when average usage is within limits.
- **Regional quota independence**: Most providers assign quotas per region per subscription/account, so geographic distribution affects capacity planning.
- **Model-specific limits**: Quotas are not uniform across models. A larger, more capable model typically carries stricter TPM and RPM ceilings.

Scope and Structure of This Paper

This paper covers the mechanics of TPM, RPM, and TPD across major cloud providers -AWS, Azure, and GCP; default and maximum quota values; a structured performance validation methodology; and prompt engineering and token optimization techniques to improve both performance and cost efficiency.

3. Understanding LLM Performance Primitives

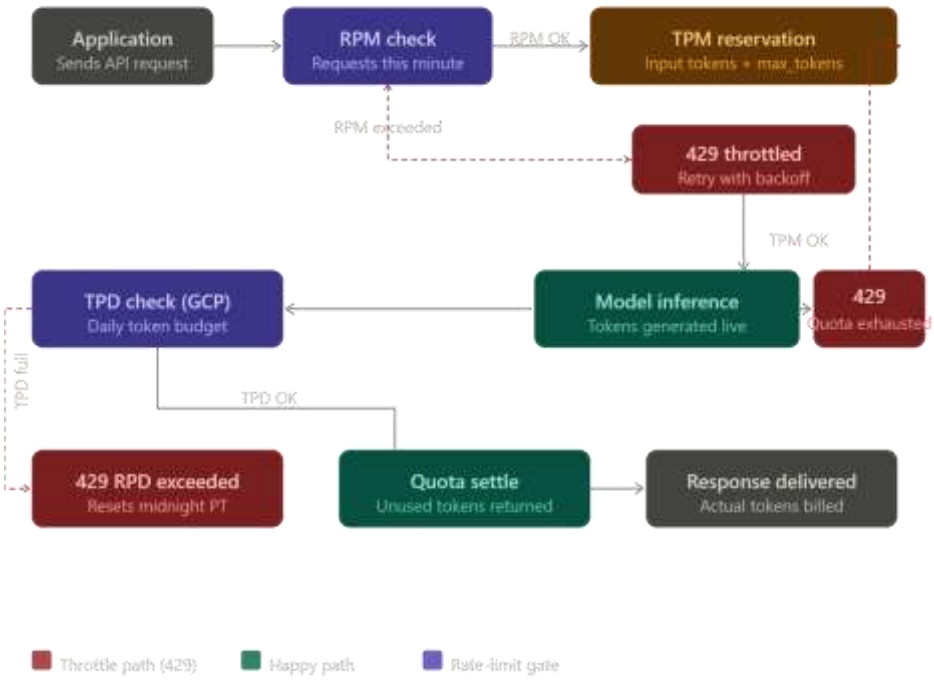
Tokens: The Unit of LLM Computation

A token is the fundamental unit of input and output for large language models. A common rule of thumb is that one token corresponds to approximately four characters of English text, or roughly three-quarters of a word. A 100-word paragraph typically consumes around 133 tokens. Code, structured data, and non-English languages often tokenize less efficiently.

Both input tokens (the prompt, system instructions, conversation history, and retrieved context) and output tokens (the model's generated response) count toward consumption. Output tokens are more expensive from a quota standpoint: for example, on AWS Bedrock, the Claude 4 series applies a burndown rate of 5x for output tokens in quota calculations.

The Token Request Lifecycle

The following diagram illustrates how a single API request traverses through each rate-limit gate before being processed and returned to the application.



TPM (Tokens Per Minute)

Tokens Per Minute is the primary capacity metric across all cloud LLM providers. It represents the total number of tokens — input plus output, with any applicable multipliers — that an account can process in a rolling one-minute window. TPM limits govern throughput and are typically enforced per-model, per-region, per-account.

RPM (Requests Per Minute)

Requests Per Minute limits the number of distinct API invocations regardless of their token size. RPM is the binding constraint for applications with many small, frequent requests. It is typically evaluated on a short rolling window — as brief as one second on Azure — making burst traffic a significant risk even for applications that appear within limits on a per-minute average.

TPD / RPD (Tokens/Requests Per Day)

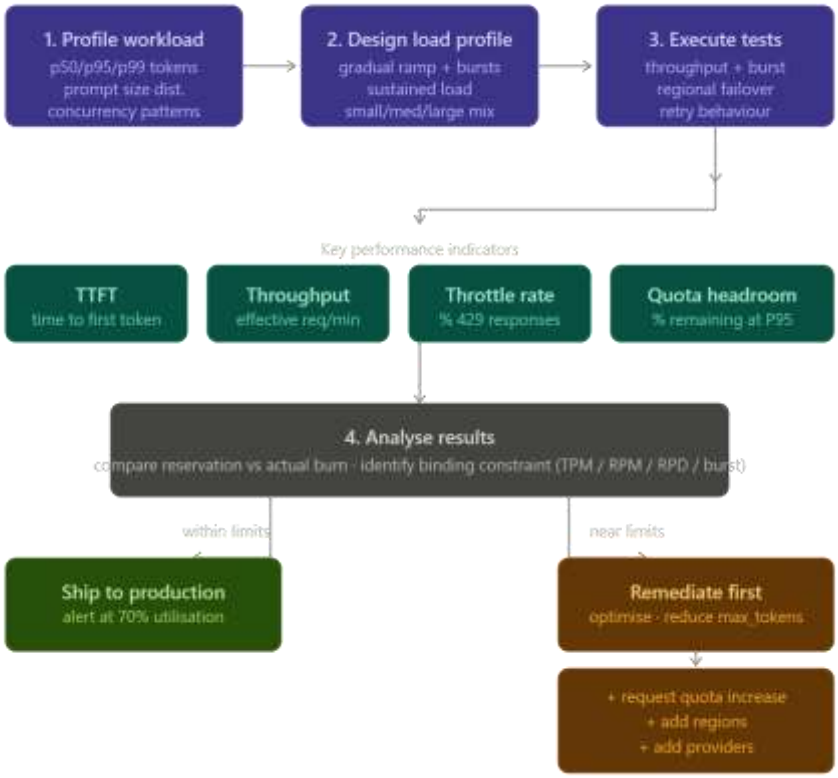
Google's Gemini API enforces daily limits in addition to per-minute limits. These daily caps reset at midnight Pacific Time. Daily limits are particularly relevant for batch processing workloads that may operate well within per-minute limits but accumulate large total volumes over 24 hours.

Soft Limits vs. Hard Limits

Soft limits (default quotas) are the starting values assigned when an account is created. They can be increased by submitting a quota increase request. **Hard limits** are the maximum values a quota can reach regardless of requests. Understanding whether a ceiling is soft or hard is critical for capacity planning: a soft ceiling can be raised through a support process, while a hard ceiling requires architectural workarounds.

4. Performance Validation Framework for LLM Applications

The following diagram maps the end-to-end performance validation workflow from workload profiling through KPI measurement to the ship/remediate decision gate.



Testing Dimensions

Throughput testing measures how many requests and tokens the application can process per minute under sustained load. **Latency testing** measures TTFT and total generation time under varying load. **Token variability testing** measures the range of input and output token counts for representative workload samples. **Burst testing** exercises behaviour when traffic spikes significantly above baseline. **Regional failover testing** validates that multi-region routing logic correctly re-routes traffic when one region is throttled.

KPIs and Load Profile Design

Performance test loads must reflect token variability and concurrency patterns of production workloads. Include variable prompt sizes (small: 50–200 tokens; medium: 500–2,000; large: 5,000+), weighted by expected production distribution. Allow responses to complete naturally across a varied prompt set to observe actual output token distribution. Include realistic inter-request think times for user-facing applications.

5. Observability: Native and Third-Party Tooling

The Three Pillars of LLM Observability

Quality metrics are the hardest of the three pillars to measure because unlike latency or token counts, there's no objective number the API returns — you must evaluate the output itself. Here's how each is typically measured:

Hallucination rate is measured by running an LLM-as-judge: a second model (often a stronger one, or a specialized evaluator like Prometheus or GPT-4o) is given the original prompt, the retrieved context, and the response, then asked to flag any claims not grounded in the source material. Tools like Arize Phoenix and Ragas have built-in hallucination scorers.

Relevance scores measure whether the response actually answered the question asked. Again, LLM-as-judge is the standard approach — the evaluator scores semantic alignment between the question and answer. Embedding cosine similarity (comparing query and response vectors) is a cheaper, faster proxy.

RAG retrieval quality is evaluated separately from the generation step. The key metrics are context precision (did we retrieve relevant chunks?) and context recall (did we miss important ones?). Ragas is the most widely used framework for this — it computes these scores automatically given the question, retrieved chunks, and ground-truth answer.

LLM-as-judge scores are the backbone of most quality pipelines. The pattern is: define a rubric (e.g. "rate this response 1–5 for helpfulness and factual accuracy"), send the rubric + response to an evaluator model, parse the structured score back. Langfuse, LangSmith, and Arize Phoenix all have prompt templates and pipelines for this built in.

Output length compliance is the simplest — just check that len(response_tokens) falls within the expected range for the task. Pure code, no model needed.

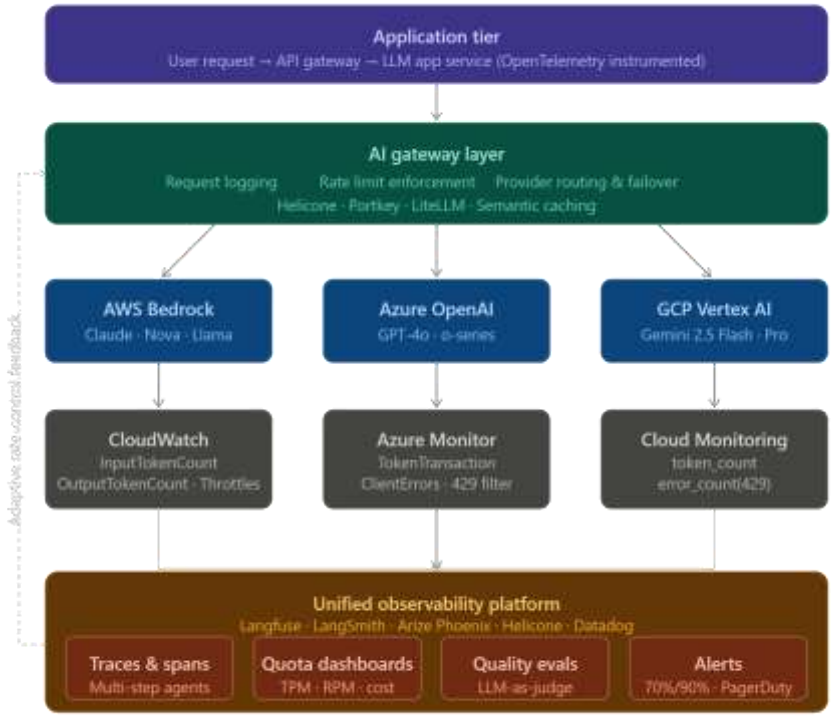
User feedback signals come from explicit thumbs up/down or star ratings in your UI, wired back into your observability platform. These are the ground truth that all the automated scores are trying to approximate.

Toxicity / safety flags are typically handled by a dedicated classifier — either your cloud provider's content filter (Azure Content Safety, AWS Bedrock Guardrails) or an open-source model like Llama Guard — running as a sidecar on every response.

The practical architecture is usually: lightweight automated scorers (embedding similarity, length checks, safety classifiers) run on 100% of traffic in near-real-time, while the heavier LLM-as-judge evaluations run on a sampled subset (5–10%) to keep costs manageable. Human review is reserved for flagged or low-scoring samples.

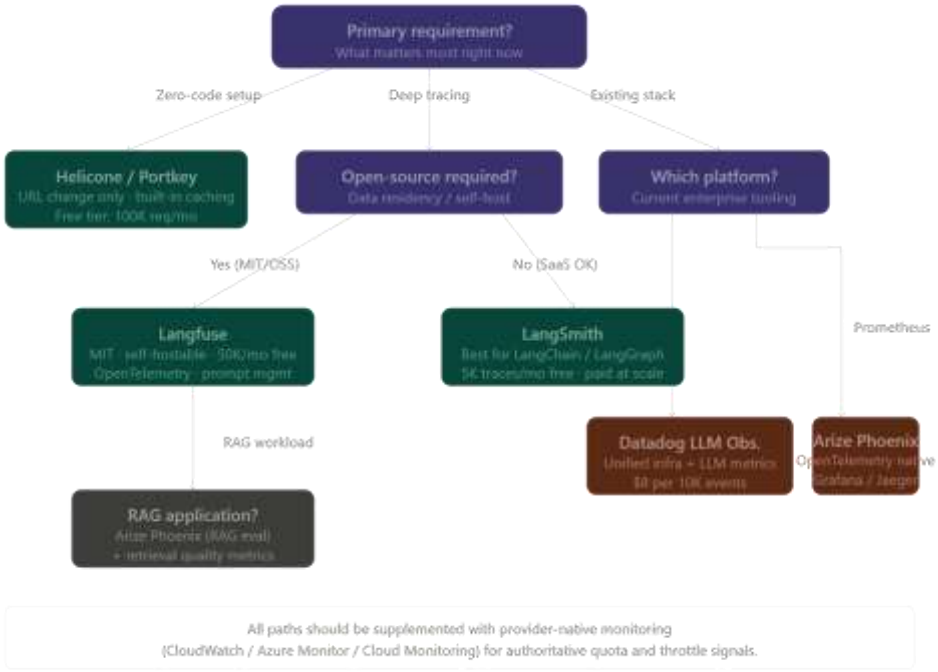
End-to-End Observability Architecture

The following reference architecture shows how signals flow from the application tier through the AI gateway and provider-native monitoring into a unified observability platform.



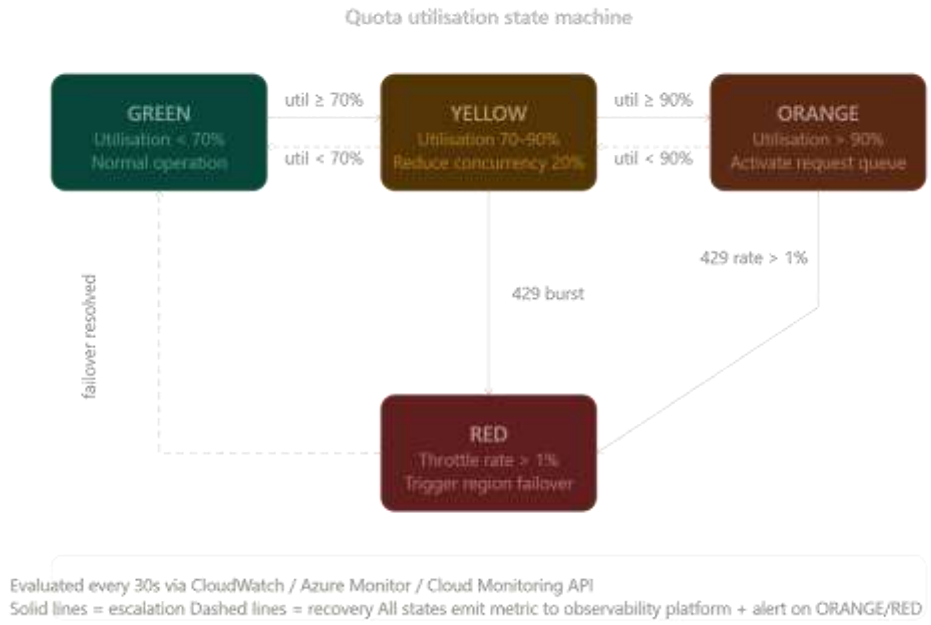
For Cloud Native Observability metrics, please refer the attached appendices for details on metrics provided by Major cloud providers

Third-Party Specialised LLM Observability Platforms



Platform	***Integration***	***Open Source***	***Free Tier***	***Best For***
Helicone	URL change only	Partial	100K req/mo	Fast setup, cost tracking, caching
Langfuse	SDK / OTel	MIT licence	50K events/mo	Self-hosting, prompt mgmt, tracing
LangSmith	SDK (native LangChain)	No	5K traces/mo	LangChain / LangGraph teams
Arize Phoenix	OTel	Elastic 2.0	Self-hosted free	RAG evaluation, OpenTelemetry stacks
Portkey	URL change / SDK	Partial	Available	Multi-provider routing, guardrails
Datadog LLM Obs.	Agent / SDK	No	Paid ($8/10K events)	Existing Datadog enterprise users

Throttle Detection and Adaptive Response Pattern



6. Prompt Engineering for Performance

Recommended Prompt Structure

The following diagram shows the optimal structural ordering of a production prompt, the token cost of each layer, and common anti-patterns that inflate token usage without improving quality.

Concise Prompting

The most direct performance optimization is eliminating unnecessary tokens from prompts. Replacing verbose prose instructions with structured, direct commands consistently reduces token counts by 30–70% for equivalent output quality. Common sources of unnecessary tokens include verbose disclaimers, restated context, repetitive instructions, and unnecessary few-shot examples.

Chain-of-Thought and Its Token Cost

Chain-of-Thought (CoT) prompting significantly improves accuracy on complex reasoning tasks but comes with a substantial token cost — CoT responses consume 3–10x more output tokens than direct-answer prompts. For performance-sensitive applications, CoT should be applied selectively, only for tasks where reasoning quality materially affects outcomes. Skeleton-of-Thought (SoT) — which generates a structured outline first and expands sections in parallel — reduces CoT latency by enabling parallel token generation.

Context Engineering and Token Budgeting

Key practices include setting explicit token budgets for each prompt component, summarizing or truncating older conversation turns rather than forwarding the entire history, and in RAG architectures limiting retrieved chunks to the top-k most semantically relevant passages.

7. Token Optimization Strategies

The Economics of Token Optimization

Token optimization reduces three costs simultaneously: API billing, quota consumption, and latency. Output tokens carry a disproportionate cost: priced 4–5x higher than input tokens by most providers and, on AWS Bedrock with the Claude 4 series, consuming 5x the quota per actual token. Limiting output length has outsized impact on both cost and throughput.

Setting max_tokens Correctly

The most impactful single optimization for Bedrock workloads is setting max_tokens to a realistic value. Organizations should profile their production response length distribution (p50, p95, p99) and set max_tokens at the p99 value with a modest buffer. This eliminates virtually all unnecessary over-reservation.

Prompt Caching

All three major providers support prompt caching, which stores computed key-value representations of frequently repeated prompt prefixes. Cache read tokens are priced at approximately 10% of standard input token cost (Anthropic). Static content should always be placed at the beginning of the prompt, as cache matching is prefix-based.

Semantic Caching

Semantic caching stores LLM responses alongside vector embeddings of user queries. When a sufficiently similar query arrives, the cached response is returned without an LLM invocation — eliminating the API call entirely. Semantic caching can reduce LLM API costs by up to 73% for applications with high query repetition.

RAG Context Optimization

Key RAG token optimization practices: limit top-k retrieval to 3–5 chunks; compress retrieved chunks to remove redundant text; use semantic chunking by paragraph/section rather than fixed character counts; enforce a fixed token ceiling on retrieved context.

Model Routing

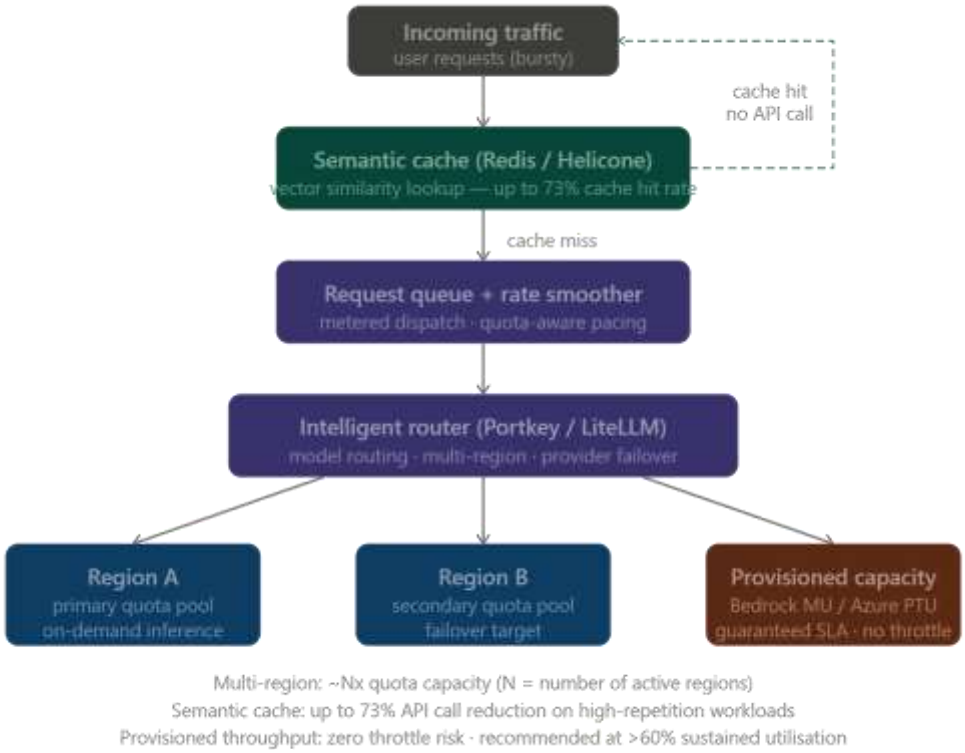
Routing requests to the smallest model that can reliably handle the task is one of the most effective architectural optimizations. Use a lightweight classifier to categorize incoming requests by complexity before routing. Organizations that adopted intelligent model selection alongside prompt optimization reported approximately 30% improvement in cost efficiency.

8. Architectural Patterns for Scaling Beyond Limits

Scaling Architecture Reference Design

The following diagram shows how the four primary scaling patterns — semantic caching, request queuing, intelligent routing, and multi-region / provisioned distribution — stack together to move a workload beyond on-demand quota ceilings.

SCALING ARCHITECTURE: FROM BURST TRAFFIC TO RELIABLE THROUGHPUT



Multi-Region Distribution

Since quotas are independent per region, a load balancer distributing traffic across three regions provides approximately 3× the throughput of a single-region deployment. Considerations include latency variation across regions, data residency requirements, and intelligent retry routing that avoids sending a retried request back to a throttled region.

Multi-Provider Fallback

An API gateway layer routes requests to the primary provider and fails over to an alternative when the primary returns sustained 429 responses. The primary challenge is model output consistency across providers, requiring applications to tolerate output variability or invest in model-specific output normalization.

Asynchronous and Batch Processing

Batch inference APIs (available on AWS Bedrock and GCP Vertex AI) process requests against a separate quota pool, allowing high-volume offline workloads to run without contending with real-time interactive traffic quotas. Separating real-time and batch workloads into independent quota allocations is a best practice for enterprise deployments.

9. Conclusion and Recommendations

Summary of Findings

LLM application performance validation is a multi-dimensional discipline spanning cloud infrastructure quotas, token consumption mechanics, prompt design, and application architecture.

AWS Bedrock's reservation-based TPM counting model is the most distinctive behaviour across providers. Setting max_tokens to the model's maximum by default is a common misconfiguration that causes unexpected throttling at seemingly low utilization rates.

Azure OpenAI's fixed 6 RPM-per-1,000-TPM coupling means throughput planning requires balancing both axes simultaneously. Applications with high-frequency, small requests may be RPM-limited well below their TPM ceiling.

Google Vertex AI's dynamic shared quota model and spend-based tier progression make quota planning a function of FinOps strategy. Teams should model their expected growth trajectory against tier thresholds and account for the December 2025 reductions to Free and Tier 1 quotas.

Observability must be treated as a first-class concern from day one. Provider-native tools (CloudWatch, Azure Monitor, Cloud Monitoring/Logs Explorer) provide the authoritative view of quota consumption; third-party platforms (Langfuse, Helicone, LangSmith, Arize Phoenix, Datadog) add application-layer tracing, quality evaluation, and cross-provider visibility that native tools lack.

Prompt and token optimization consistently yields 30–70% reductions in token consumption across diverse application types.

Recommendations by Maturity Stage

For teams in early development and prototyping:

- Enable billing immediately on GCP to access Tier 1 quotas
- Set explicit, realistic max_tokens values from the first line of LLM code
- Add Helicone or Langfuse in the first week — observability debt compounds quickly
- Establish CloudWatch / Azure Monitor / Cloud Monitoring dashboards before first performance test

For teams preparing for production:

- Profile production prompt token distributions (p50, p95, p99) before selecting provisioned capacity
- Implement exponential backoff with jitter for all LLM API calls
- Conduct burst testing at 2–5× expected peak load
- Submit quota increase requests 4–6 weeks before launch
- Configure throttle alerts at 70% and 90% utilisation with actionable runbooks

For teams operating at scale:

- Implement multi-region routing for resilience and throughput multiplication
- Evaluate provisioned throughput for sustained high-utilization workloads
- Adopt semantic caching for applications with significant query repetition
- Separate real-time and batch workloads into independent quota pools
- Build adaptive client-side rate limiting with real-time quota utilization feedback
- Graduate to enterprise observability (Datadog, Arize AX) for unified infra + LLM correlated alerting

Looking Ahead

Cloud LLM quota structures are evolving rapidly. Engineering teams should treat quota limits as a dynamic operational parameter — not a fixed architectural constant — and build observability, alerting, and adaptive capacity mechanisms that respond to quota changes without requiring application rewrites. The organizations best positioned for continued

scale will be those that treat token efficiency and observability as first-class engineering concerns from the earliest stages of application development.

Disclaimer: This whitepaper reflects publicly available provider documentation and community findings as of March 2026. Specific quota values change frequently; always verify current limits in your provider's official documentation and console.

References

- AWS Documentation: Quotas for Amazon Bedrock — *docs.aws.amazon.com/bedrock/latest/userguide/quotas.html*
- AWS Documentation: How tokens are counted in Amazon Bedrock — *docs.aws.amazon.com/bedrock/latest/userguide/quotas-token-burndown.html*
- Microsoft Learn: Azure OpenAI Quotas and Limits — *learn.microsoft.com/en-us/azure/foundry/openai/quotas-limits*
- Microsoft Learn: Manage Azure OpenAI Quota — *learn.microsoft.com/en-us/azure/ai-services/openai/how-to/quota*
- Google Cloud: Generative AI on Vertex AI Quotas — *cloud.google.com/vertex-ai/generative-ai/docs/quotas*
- Google Cloud: Standard PayGo for Vertex AI — *cloud.google.com/vertex-ai/generative-ai/docs/standard-paygo*
- Google AI: Gemini API Rate Limits — *ai.google.dev/gemini-api/docs/rate-limits*
- Avahi AI: Performance Testing AWS Bedrock Foundational Models (January 2026)
- AWS re:Post: TPM & RPM Quota Monitoring Dashboard for Amazon Bedrock (January 2026)
- Helicone: The Complete Guide to LLM Observability Platforms (2025)
- ZenML: 10 Best LLM Monitoring Tools (2025)
- Langfuse: Open-source LLM observability — langfuse.com
- Arize AI: Phoenix Open-Source LLM Observability — phoenix.arize.com
- Anthropic Prompt Engineering Documentation — docs.anthropic.com

10. Appendices

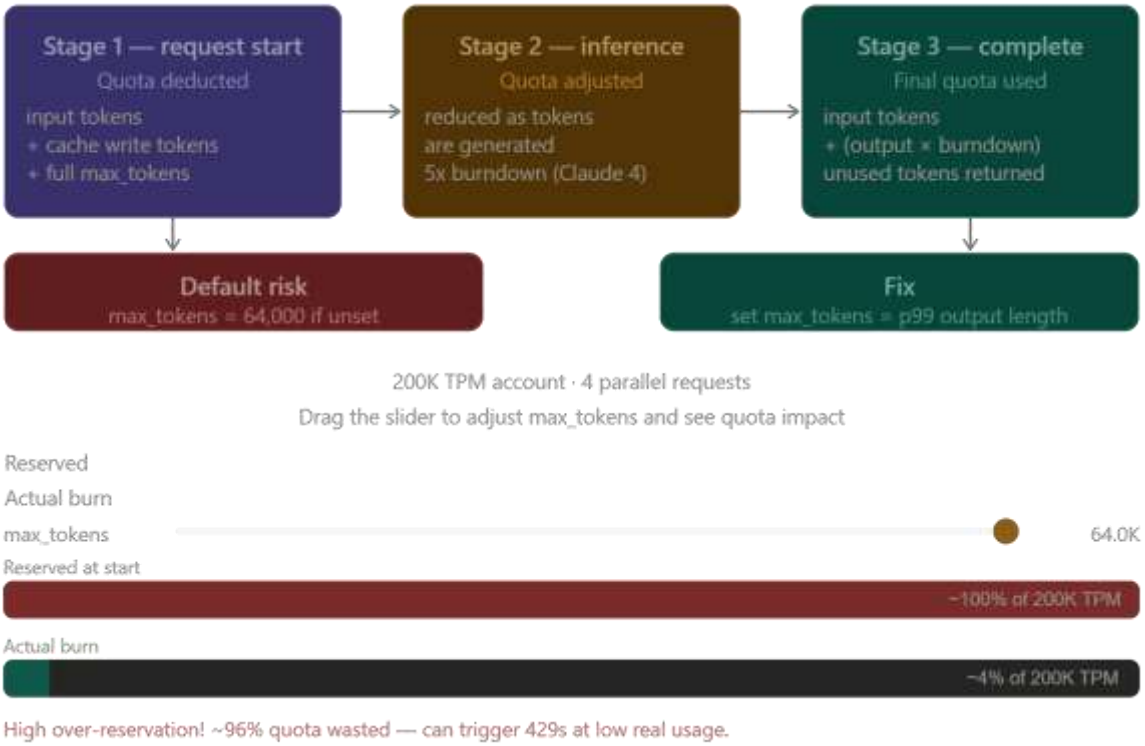
AWS Bedrock: Rate Limits, Quotas, and Regional Behaviour

Overview of Bedrock's Quota Model

Amazon Bedrock provides access to foundation models from Anthropic (Claude family), Meta (Llama), Amazon (Nova, Titan), Mistral, and others. Quotas are enforced at the account level, per model, per AWS region. New AWS accounts receive significantly lower default quotas — sometimes as low as 2–3 RPM on newer Claude models — while established accounts may have 250 RPM or more on the same models.

The Token Reservation Model

AWS Bedrock uses a three-stage token counting process that is the most common source of unexpected throttling in production.



Default and Elevated Quotas

Metric	*New Account*	*Standard Account*	*Elevated (Special Request)*
RPM (Claude Sonnet)	2–3	250–800	Up to 6,000+
TPM (Claude Sonnet)	Very low	200,000–600,000	Up to 36,000,000
RPM (Claude Opus)	2	50–100	By request
RPM (LLaMA 3.3 70B)	Low	~800	Up to 6,000 (tested)
TPM (LLaMA 3.3 70B)	Low	~600,000	Up to 36,000,000 (tested)

Provisioned Throughput

For workloads requiring consistent, guaranteed throughput without throttling, AWS Bedrock offers Provisioned Throughput — a reserved-capacity model where customers purchase model units (MUs) that guarantee a defined TPM regardless of on-demand system load. Provisioned throughput is billed continuously and is appropriate for latency-sensitive production workloads with predictable traffic patterns.

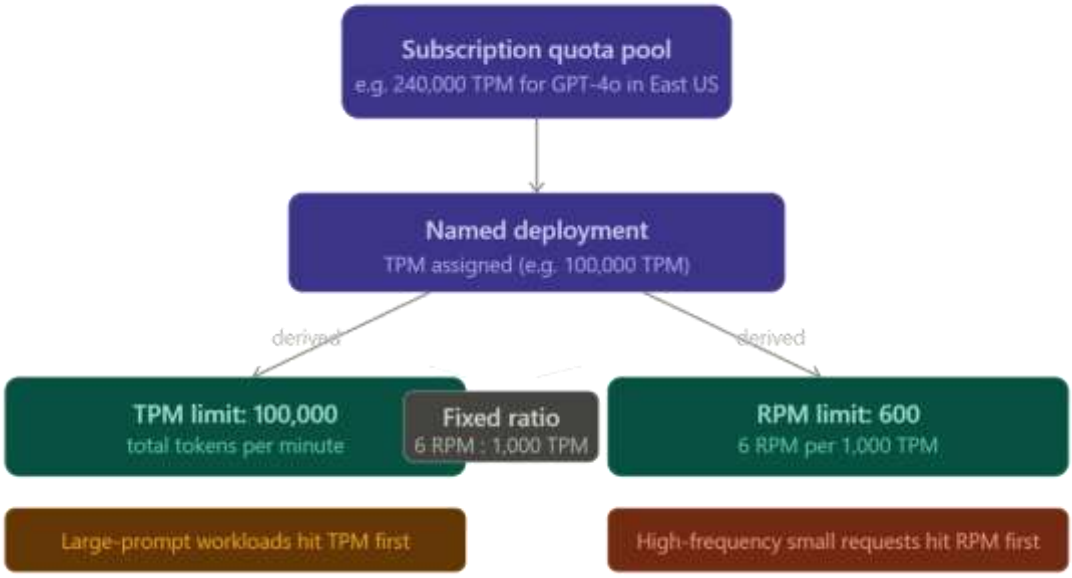
Cross-Region Inference and Quota Monitoring

AWS Bedrock supports cross-region inference, which allows requests to be routed across multiple AWS regions automatically, effectively pooling quotas. CloudWatch metrics (InputTokenCount, OutputTokenCount, InvocationThrottles) provide the primary quota observability surface. Setting max_tokens to realistic values — profiled from production p99 response lengths — is the single highest-impact optimization for Bedrock workloads.

Azure OpenAI Service: Rate Limits, Quotas, and Regional Behaviour

Overview and the TPM-to-RPM Ratio

Azure OpenAI enforces a fixed relationship between TPM and RPM: **6 RPM per 1,000 TPM**. This ratio is applied automatically when a deployment is created. The following diagram shows how a single quota allocation translates to both axes and where different workload types hit their ceiling first.



Default Quota Values

Model	***Subscription Type***	***Default TPM***	***Derived RPM***	***Max TPM (typical)***
GPT-4o	Pay-As-You-Go	Varies by region	6 per 1K TPM	240,000–450,000
GPT-4o (Global Standard)	Enterprise	5,000,000	30,000	5,000,000+
GPT-4 series	CSP/MSDN	8,000	48	8,000 (hard limit)
GPT-3.5 Turbo	Standard	240,000	1,440	240,000 per region

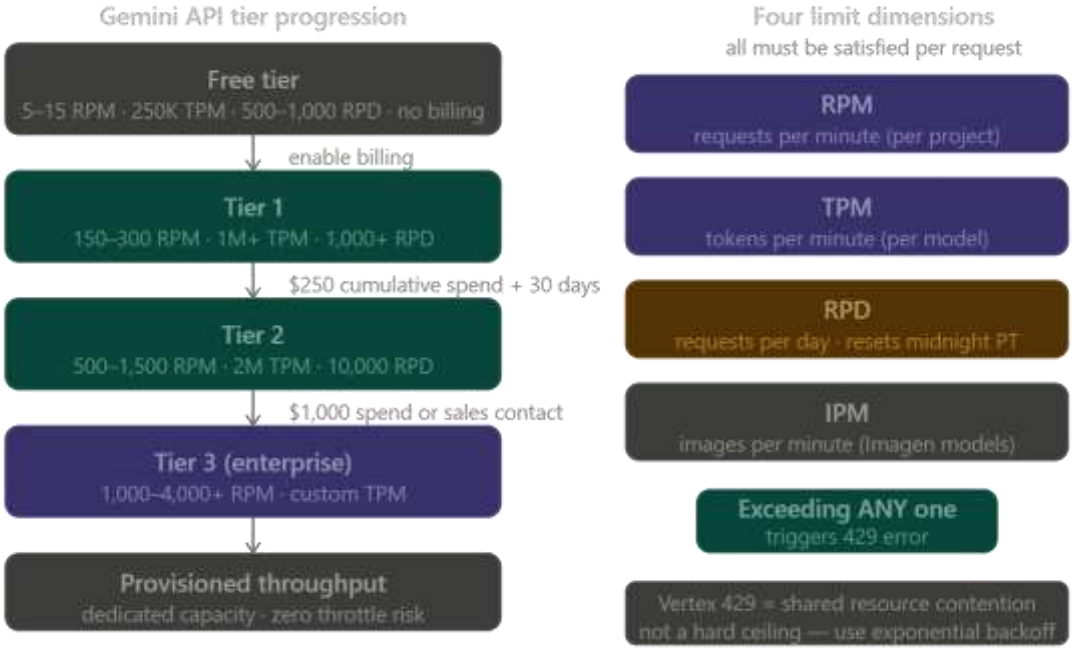
Multi-Region Strategy and PTU

Azure OpenAI quotas are allocated per region per subscription independently. Deploying the same model in East US and West US gives the subscription two separate quota pools, enabling aggregate throughput higher than any single region's quota. For mission-critical workloads, Azure offers Provisioned Throughput Units (PTU) — dedicated, reserved compute capacity purchased as a monthly commitment. The break-even for PTU vs. TPM billing typically occurs when sustained utilization exceeds 60–70% of provisioned capacity.

Google Cloud Vertex AI & Gemini API: Rate Limits, Quotas, and Tier Structure

Tier Progression and Four-Dimensional Rate Limiting

Google structures access through spend-based progressive tiers, and unlike AWS and Azure enforces limits on **four simultaneous axes**. Exceeding any single dimension triggers a 429 independently of the others.



System limit: 30,000 RPM per model per region (all tiers)

TPM tiers are independent per model family

Standard PayGo

Vertex AI's Standard PayGo introduces a spend-based dynamic quota model. An organization's baseline throughput capacity scales automatically based on its rolling 30-day total spend across eligible Vertex AI services. Higher-spending organizations are automatically promoted to higher tiers, providing greater access to shared compute resources.

Cross-Provider Comparison and Regional Considerations

Comparative Quota Architecture

Feature	***AWS Bedrock***	***Azure OpenAI***	***GCP Vertex AI / Gemini***
Primary limits	TPM, RPM	TPM, RPM	TPM, RPM, RPD, IPM
Quota scope	Per account, per model, per region	Per subscription, per model, per region	Per project / org, per model, per region
RPM/TPM coupling	Independent	Fixed: 6 RPM per 1K TPM	No separate RPM tier limit (30K system cap)
Output token multiplier	1x–5x (model-dependent)	Not applied	Not applied
New account defaults	Very low (2–3 RPM)	Subscription-type dependent	Free tier; billing required for growth
Provisioned capacity	Yes (Model Units)	Yes (PTU/PTU-M)	Yes (Provisioned Throughput)
Quota increase path	Service Quotas console / support	Azure portal / quota request form	Google Cloud console / account team

Default retry guidance	Exponential backoff	Exponential backoff	Exponential backoff + jitter
Daily limits	No	No	Yes (RPD on Gemini API)

Throttling Error Codes

Provider	***HTTP Status***	***Error Type***	***Recommended Response***
AWS Bedrock	429	ThrottlingException	Exponential backoff, reduce parallelism
Azure OpenAI	429	RateLimitError	Exponential backoff, check RPM window
GCP Vertex AI	429	RESOURCE_EXHAUSTED	Exponential backoff + jitter; not a hard cap

Provider-Native Observability Tools

AWS CloudWatch for Amazon Bedrock

Key CloudWatch metrics:

METRIC NAMESPACE: AWS/Bedrock

Metric Name	**Description**
InputTokenCount	**Tokens consumed in prompts per request**
OutputTokenCount	Tokens generated per request
InvocationLatency	**End-to-end response time (ms)**
FirstByteLatency	Time-to-first-token for streaming
InvocationThrottles	**Count of throttled requests (429s)**
ModelInvocations	Total request count (maps to RPM)

DERIVED QUOTA METRICS:

TPM Reservation = InputTokenCount + MaxTokens (per request)

TPM Actual Burn = InputTokenCount + (OutputTokenCount × BurndownRate)

Quota Headroom % = (1 - TPM_Reservation_1min / AccountTPMLimit) × 100

Recommended CloudWatch alarms:

Alarm	**Threshold**	**Action**
InvocationThrottles > 0	Any throttle event	PagerDuty / SNS alert
TPM Utilisation > 70%	Rolling 5-min average	Warning notification
TPM Utilisation > 90%	Rolling 5-min average	Critical — trigger quota increase

InvocationLatency P99 SLA	> Model-specific SLA	Engineering alert

Azure Monitor for Azure OpenAI

Key Azure Monitor metrics:

RESOURCE PROVIDER: Microsoft.CognitiveServices/accounts

Metric	Description
TokenTransaction	Total tokens processed (input + output)
ProcessedPromptTokens	**Input token count per invocation**
GeneratedCompletionTokens	Output token count per invocation
TotalRequests	**Request count (maps to RPM consumption)**
ClientErrors	4xx responses (includes 429 throttles)
LatencyMs	**End-to-end request latency**
TimeToResponse	Time-to-first-token for streaming

APIM LLM Token Limit Policy (per-consumer enforcement):

```
<azure-openai-token-limit
  counter-key="@(context.Subscription.Id)"
  tokens-per-minute="50000"
  estimate-prompt-tokens="true"
  remaining-tokens-header-name="x-ratelimit-remaining-tokens"
  remaining-requests-header-name="x-ratelimit-remaining-requests" />
```

Google Cloud Monitoring and Logs Explorer

Key Cloud Monitoring metrics:

METRIC PREFIX: aiplatform.googleapis.com

Metric	Description
prediction/online/token_count	Token throughput by model
prediction/online/request_count	Request rate per model
prediction/online/response_latencies	Latency distribution
prediction/online/error_count	Errors by code (incl. 429)

quota/generate_content_token_count/...	Tokens vs. quota limit

Logs Explorer query examples:

-- Find all throttled requests in the last hour

resource.type="aiplatform.googleapis.com/Endpoint"

severity=ERROR

httpRequest.status=429

-- Token consumption by model over time

resource.type="aiplatform.googleapis.com/Endpoint"

jsonPayload.model_id=~"gemini-."*

| summarize sum(jsonPayload.token_count) by model_id, bin(timestamp, 1m)