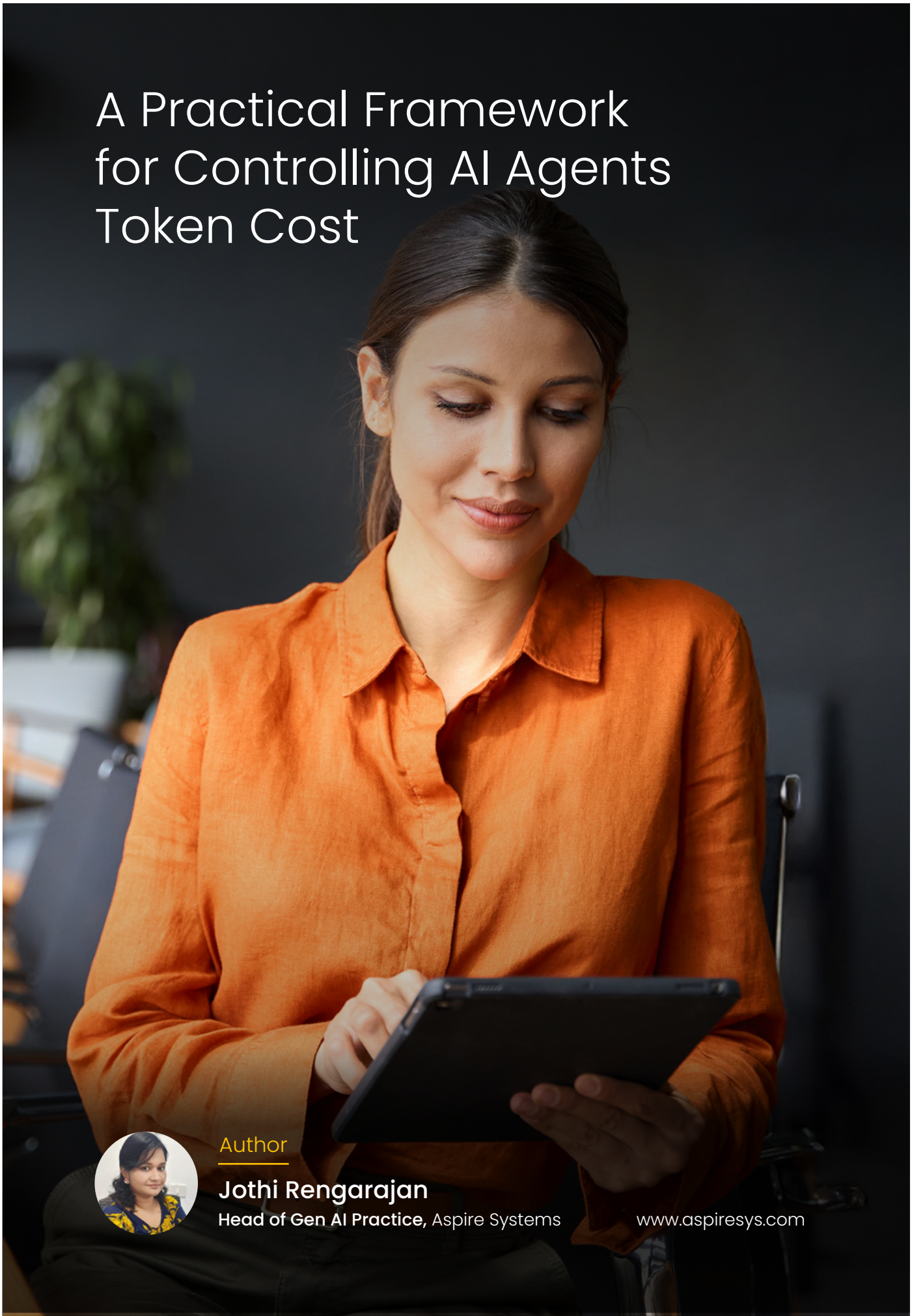
aspire SYSTEMS

Whitepaper

A Practical Framework for Controlling AI Agents Token Cost



Author

Jothi Rengarajan

Head of Gen AI Practice, Aspire Systems

www.aspiresys.com

AI cost is no longer proportional to user traffic

Autonomous and semi-autonomous AI agents; systems that plan, invoke tools, iterate, and delegate work to sub-agents, consume tokens very differently from traditional chat-based LLM applications. A single business task can trigger dozens of model calls, each carrying an expanding context window, automatic retries, and parallel execution across multiple agents. Without appropriate controls, what begins as a low-cost interaction can quickly become a multi-dollar workflow, turning predictable AI spend into a volatile and difficult-to-govern cost center.

This whitepaper presents an in-depth model for controlling that cost, organized around four levers that compound:

Application layer

Architectural and prompt token controls inside the agent code itself.

FinOps layer

Applying established FinOps principles to allocate, forecast, optimize, and govern AI spend, just as organizations already do for cloud infrastructure.

Proxy/gateway layer

A centralized control point between all agents and model providers (Azure OpenAI, AWS Bedrock, Anthropic, etc.) that enforces governance without requiring changes to agent code.

Tooling integration

Extending existing observability, gateway, and cost management platforms to provide end-to-end visibility into AI consumption.

No single layer, on its own, is sufficient. Application-layer controls reduce waste within individual agents but provide limited cross-team visibility. Gateways enforce centralized policies but cannot fully understand an agent's internal reasoning. FinOps relies on clean, tagged data from both layers to produce a trustworthy cost model.

The recommendation is to implement all four in sequence, starting with the layer closest to the code (cheapest, fastest ROI) and working outward to governance (slowest, highest leverage at scale).

AI Agents Change the Economics of LLM Consumption

Traditional API cost governance assumes a straightforward relationship between user requests and model consumption: one request leads to one model call, the context window remains relatively small, and costs scale predictably with usage.

Agentic AI breaks each of these assumptions. Unlike a conventional LLM interaction, an AI agent plans its approach, invokes tools, reflects on intermediate results, retries failed operations, and may delegate work to specialized sub-agents. Every step generates additional model calls, expands the context window, and increases token consumption.

The difference is illustrated below:

Traditional LLM call	Agentic workload
One request → one response	One task → multiple model calls (planning, tool calls, reflection, sub-agent delegation)
Context window is limited to the prompt	Context grows continuously through tool outputs, prior messages, and retrieved documents
Retries are rare, user-initiated	Retries are automatic (tool failures, JSON parsing issues, timeouts) and often invisible to users
One caller, one bill	Multiple agents and services invoke models through shared APIs
Cost scales roughly with usage	Cost scales with model selection, context length, number of reasoning steps, and parallel fan-out

The result is what many FinOps practitioners describe as the 'agent cost multiplier.' The same business task can vary by 10–100x in token cost depending on how the agent is designed, with limited visibility into which architectural decisions are driving the increase.

This is why cost optimization cannot be treated as a post-deployment exercise. It must be designed into the application architecture from the outset.

Application-Layer Controls: Where Cost Optimization Begins

The application layer provides the earliest, and often the most effective, opportunity to control token consumption because it has complete visibility into why every model call is made.

1. Context management

Selective tool input and output inclusion:

Truncate, summarize, or generate diffs for large tool outputs such as logs, API responses, or file contents before adding them back into the context window. For example, we have implemented tools that pass an Azure Blob Storage or Amazon S3 document path instead of the document itself. The document is then retrieved and optimized before processing, significantly reducing token consumption.

Prompt caching:

Structure prompts so that the static portion—system prompts, tool schemas, and few-shot examples—remains unchanged, while dynamic content is appended at the end. Azure OpenAI, Anthropic, and AWS Bedrock all support prompt or context caching, reducing the cost of cached tokens by 80–90%. Prompt design directly influences how effectively these caching mechanisms work.

2. Model routing and tiering

Route by task complexity

Use smaller, faster models (for example, Haiku-class or GPT-4o-mini-class) for simple tasks such as classification and routing, reserving frontier models only for reasoning-intensive steps.

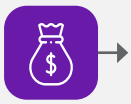
Route by reasoning effort, not just model family

Where supported, configure per-call reasoning levels (low, medium, or high), as many agent tasks such as formatting and simple lookups do not require high-effort reasoning.

Configure models by agent type

Treat model and reasoning effort as per-agent configurations rather than global defaults, so a ticket classification agent and a root cause analysis agent are not forced into the same cost profile.

3. Budget enforcement and execution limits



Assign explicit budgets to every agent run:

Every agent should operate within a token or dollar budget object, rather than relying solely on maximum iteration counts. Loop-until-done patterns without budget ceilings are a common cause of runaway costs.

Limit execution depth:

Enforce maximum tool-call depth and sub-agent fan-out for each task, with a defined "insufficient budget, returning partial result" exit path instead of silently truncating execution.

Fail fast on repeated errors:

Avoid indefinite retry loops for malformed tool calls. Cap retries (typically two or three attempts) with exponential backoff, surface failures appropriately, and avoid repeatedly re-prompting the model.

4. Concurrency and rate limiting

Parallel execution improves performance but can also multiply costs unexpectedly. Application-level controls should cap concurrent sub-agent execution based on available compute resources or provider quotas. This prevents a single task from generating hundreds of simultaneous model calls that increase both token consumption and the risk of provider throttling.

5. Per-call instrumentation

Cost optimization depends on accurate telemetry. Every model invocation should capture:

- Input, output, and cached token counts
- Latency
- Model metadata
- Cost per call

This data should be emitted through an LLM-specific observability layer such as LLMmetry and tagged with metadata including:

- Agent type
- Task ID
- User ID (where applicable)
- Team or project
- Model

This creates the task-level usage record required by every downstream capability, including gateway analytics, FinOps dashboards, chargeback, showback, and anomaly detection. Instrumentation should be implemented as close to the application as possible rather than reconstructed later from provider billing reports.

Centralizing Governance Through an LLM Gateway

A proxy (LLM gateway) positioned between AI agents and model providers centralizes policy enforcement, regardless of the team, application, or provider being used.

What an LLM gateway provides beyond application-layer controls

 **Consistent policy enforcement:**	Apply budgets, retries, caching, and governance policies uniformly across all agents without requiring each team to implement them independently.
 **Cross-provider abstraction:**	Present a single API layer across Azure OpenAI, AWS Bedrock, Anthropic, and OpenAI, allowing routing and failover decisions to be managed centrally rather than within each agent.
 **Infrastructure-level response caching:**	Use exact and semantic caching to identify duplicate requests across agents and teams, reducing token consumption beyond what individual applications can achieve.
 **Centralized quotas and rate limits:**	Enforce per-key or per-team spending limits and throughput quotas before requests reach the model provider, protecting both budgets and shared capacity.
 **Unified spend tracking:**	Capture token usage and costs by API key, team, or project, providing the data required for FinOps reporting, chargeback, and showback.
 **Centralized guardrails:**	Apply PII redaction, content filtering, and governance policies consistently, independent of the quality or maturity of individual agent implementations.

Reference gateway options

Tool	Type	Notable for cost control
LiteLLM Proxy	Open source, self-hosted	Per-key and per-team budgets, spend tracking, model routing and fallback, response caching, Prometheus metrics, broad Azure OpenAI, Bedrock, and Anthropic support
Portkey	SaaS/self-hosted gateway	Semantic caching, budget limits with hard cutoffs, virtual keys per team, fallback chains, cost analytics dashboard
Kong AI Gateway / Envoy AI Gateway	Enterprise API gateway w/ AI plugins	Suitable for organizations already using Kong or Envoy for API management. Supports token rate limiting and provider failover plugins
Azure API Management (APIM)- AI Gateway	Cloud-native (Azure)	Native Azure OpenAI policies including token-limit-per-key, semantic caching, and backend load balancing across PTU and pay-as-you-go deployments
Amazon Bedrock + API Gateway/ App Mesh	Cloud-native (AWS)	Bedrock Guardrails, cost allocation tags on invocations, API Gateway usage plans, and quota enforcement for Bedrock endpoints

Recommended architecture

Route all AI agent traffic through a single internal gateway endpoint, regardless of cloud provider or model. Instead of calling Azure OpenAI or AWS Bedrock directly, agents communicate with the gateway, which manages routing, retries, failover, caching, and quota enforcement before forwarding requests to the appropriate provider. This enables organizations to cap a team's daily spend, stop a runaway agent, or update routing policies through a centralized configuration change rather than modifying code across multiple repositories.

Applying FinOps Principles to AI Consumption

Application controls reduce waste. Gateways provide centralized governance. FinOps extends these capabilities by answering a broader business question: how much value is being generated for every dollar spent on AI?

The core principles remain the same as traditional cloud FinOps. The difference is that AI costs are driven by token consumption rather than infrastructure usage, and the unit of work is often an agent execution rather than a persistent compute resource.

Inform: Build visibility through cost allocation

The first step is making AI consumption visible and attributable. Every model invocation should be tagged at either the application or gateway layer with metadata such as:

- Team
- Project
- Agent type
- Environment (development, testing, production)
- Business task or ticket ID

This allows organizations to associate AI costs with business outcomes. Instead of reporting only total AI spend, teams can measure metrics such as:

- Cost per resolved incident
- Cost per pull request
- Cost per customer request
- Cost per agent execution

These tags should feed into Azure Cost Management and Cost Analysis through Azure OpenAI resource tags, and into AWS Cost Explorer and Cost and Usage Reports (CUR) through Bedrock cost allocation tags. This ensures AI spending appears within the same financial reporting framework already used for cloud infrastructure.

Tracking unit economics is equally important. Rising AI spend is not necessarily a concern if cost per business outcome continues to decrease while adoption increases.

Optimize: Use data to improve efficiency

Once AI costs are visible, optimization becomes targeted instead of reactive. Tagged usage data helps identify the largest contributors to spend. In many cases, these are not individual users, but specific agent types, particular workflow patterns, or retry loops that consume disproportionately more tokens than expected. Infrastructure commitments should also be based on observed utilization.

- **Azure OpenAI Provisioned Throughput Units (PTUs):** PTUs reduce per-token costs for consistently high workloads but may increase costs if utilization remains low or highly variable.
- **AWS Bedrock Provisioned Throughput:** Similar considerations apply. Provisioned capacity works best for predictable workloads and should be complemented with on-demand capacity for traffic spikes.

Optimization should also become part of the engineering process. Practices such as prompt caching, model routing, and context optimization should be incorporated into architecture reviews and code review checklists rather than treated as one-time improvements.

Operate: Govern AI spend continuously

Like any cloud resource, AI consumption requires ongoing governance. Operational controls should include:

- **Budget alerts and hard limits:** Azure Budgets, Azure Action Groups, AWS Budgets, and Amazon SNS can be integrated with gateway quotas so that budget thresholds trigger automated throttling instead of generating alerts after costs have already been incurred.
- **Cost anomaly detection:** Azure Cost Management anomaly detection and AWS Cost Anomaly Detection should be configured for AI workloads. Since AI traffic is naturally bursty, thresholds should be calibrated differently from traditional infrastructure alerts while remaining sensitive enough to detect runaway execution loops.
- **Chargeback and showback:** Monthly AI cost reports should be organized using the same allocation tags defined earlier and shared with engineering leaders alongside existing infrastructure chargeback reports.
- **Standardized cost reporting:** Where possible, align tagging strategies with the FinOps Foundation's FOCUS (FinOps Open Cost and Usage Specification). As AI cost reporting matures across cloud providers, adopting standardized tagging today reduces future migration effort.

Together, these practices move AI cost management from reactive monitoring to ongoing operational governance.

Integrating with Enterprise Tooling

Cost governance for AI agents does not require building a new platform. Most organizations already have observability, API management, and FinOps tooling in place. The objective is to extend these investments to include AI workloads rather than creating a parallel ecosystem.

FinOps and cost management platforms

Organizations with significant AI spend can integrate AI usage into existing multi-cloud cost management platforms.

- **CloudZero, Vantage, and Finout:** Multi-cloud cost allocation platforms that are adding AI and LLM cost visibility. These are well suited for organizations where AI has become a meaningful component of cloud spend.
- **Azure Cost Management + Power BI:** A practical option for Azure-first environments. Cost data can be exported and combined with application or gateway telemetry to build dashboards for unit economics and cost allocation.
- **AWS Cost Explorer + Cost and Usage Reports (CUR) + QuickSight/Athena:** Suitable for AWS-centric environments. CUR data, combined with gateway-generated tags, enables task-level cost attribution and detailed reporting.

Native cloud governance

Both Azure and AWS provide native capabilities that complement application and gateway controls.

Azure

- Azure OpenAI content filters
- Azure API Management token-limit policies
- Azure Monitor for latency and error correlation
- Azure Budgets for spend thresholds and hard caps

AWS

- Bedrock Guardrails
- Bedrock cost allocation tags
- Amazon CloudWatch for latency and error correlation
- AWS Budgets and AWS Cost Anomaly Detection

Bringing the layers together

When application instrumentation, gateway policies, and cloud-native cost management are integrated, organizations gain end-to-end visibility into AI consumption.

This provides:

- Task-level token and cost visibility
- Centralized governance and policy enforcement
- Business-level cost allocation
- Chargeback and showback reporting
- Cost optimization based on measurable business outcomes

The result is a governance model that builds on existing enterprise tooling instead of introducing another operational platform.

A Practical Maturity Roadmap

Organizations can implement these optimizations in phases, establishing visibility first, introducing governance next, and maturing toward enterprise-scale cost management over time.

Stage	App layer	Proxy layer	FinOps layer
Crawl	Add token and cost logging for every model call. Cap retries and maximum reasoning loops	Direct SDK calls. No centralized gateway	Review provider billing manually and establish baseline AI spend
Walk	Introduce context summarization, prompt caching, and model tiering	Route all traffic through LiteLLM, Portkey, or a similar gateway	Tag AI workloads, integrate with native cloud cost tools, and configure budget alerts
Run	Implement per-agent budgets, hard execution limits, and structured-output agents	Enforce organization-wide quotas, semantic caching, routing policies, and automated failover	Enable chargeback, optimize provisioned throughput using utilization data, and align reporting with the FOCUS specification

The maturity model is intentionally incremental. Each stage builds on the previous one, allowing organizations to realize immediate cost savings while laying the foundation for enterprise-wide governance.

Conclusion: Treat Token Cost as an Architectural Metric

For AI agents, token cost is not simply a billing metric. It is an architectural characteristic that should be designed, measured, and governed alongside reliability, latency, and accuracy.

Application-layer controls reduce unnecessary token consumption at the source. An LLM gateway provides centralized policy enforcement, routing, and organization-wide visibility. FinOps connects technical usage data with business outcomes, enabling engineering and finance teams to evaluate AI investments using a common set of metrics.

Skipping any one of these layers creates gaps. Without application-level optimization, inefficient agent design drives unnecessary spend. Without a gateway, governance depends on individual development teams implementing controls consistently. Without FinOps, organizations can measure token consumption but struggle to answer whether that spend is delivering business value.

The most effective approach is a layered one. Optimize token consumption within the application, enforce governance through a centralized gateway, and integrate AI costs into existing FinOps processes. Together, these practices shift the conversation from cost per token to cost per outcome, enabling AI spending to be forecast, governed, and optimized with the same discipline applied to every other cloud investment.

About Aspire Systems

Aspire Systems is a global technology services firm serving as a trusted technology partner for our customers. We work with some of the world's most innovative enterprises and independent software vendors, helping them leverage technology and outsourcing in our specific areas of expertise. Our core philosophy of "Attention. Always." communicates our belief in lavishing care and attention on our customer and employees.

For more info contact:
info@aspiresys.com or visit www.aspiresys.com