Building the Control Layer for Agentic AI with AI Gateway and MCP Servers

Abhishek Choudhary
(CTO, TrueFoundry)

in Collaboration with
Greg Coquillo
(AI product lead, LinkedIn TopVoice)

truefoundry

Preface

As the AI landscape evolves at an unprecedented pace, enterprise adoption of Large Language Models (LLMs) has reached an inflection point. Yet, with this rapid integration comes a web of operational challenges—API fragmentation, inconsistent latency, throttling unpredictability, opaque cost structures, and an absence of robust governance. These are not marginal issues; they are existential bottlenecks that prevent AI from moving beyond experimentation to scalable, production-grade infrastructure.

This ebook introduces the concept of the AI Gateway—a new architectural layer designed to unify, govern, and optimize AI workloads across organizations. Unlike traditional API Gateways, which are built for deterministic, schema-bound services, AI Gateways are fundamentally semantic-aware, context-sensitive, and capable of mediating between human language and machine execution.

An AI Gateway does more than route traffic. It abstracts a constantly changing set of model APIs, enforces semantic guardrails, orchestrates multi-model fallbacks, and captures fine-grained observability metrics—from token usage to model cost. It serves as the policy-enforced access layer to both LLMs and enterprise tools, integrating seamlessly with identity providers, role-based controls, and internal systems of record.

Through this book, we aim to arm platform engineers, MLOps teams, and technical architects with a deep understanding of why AI Gateways are not just useful, but foundational. You will explore architectural patterns, evaluate tradeoffs between building and buying, and understand how modern gateways integrate with emerging protocols like MCP (Model Context Protocol) and A2A (Agent-to-Agent communication).

Whether you're operating in a highly regulated environment or building globally distributed AI-native products, this guide is a practical and technical roadmap to designing and scaling your AI infrastructure with confidence

Contents

Chapter 1

The Necessity of an AI Gateway

As organizations integrate LLMs into production systems, they encounter an increasingly fragmented and unstable ecosystem. Disparate APIs, inconsistent performance, complex cost structures, and weak governance mechanisms present critical risks to operational reliability and scalability. An AI Gateway serves as the central abstraction layer that addresses these challenges systematically.

1.1 API Inconsistency Across Model Providers

Each third-party model provider exposes its own unique API interface, differing in parameter names, configuration ranges, and supported features. For example:

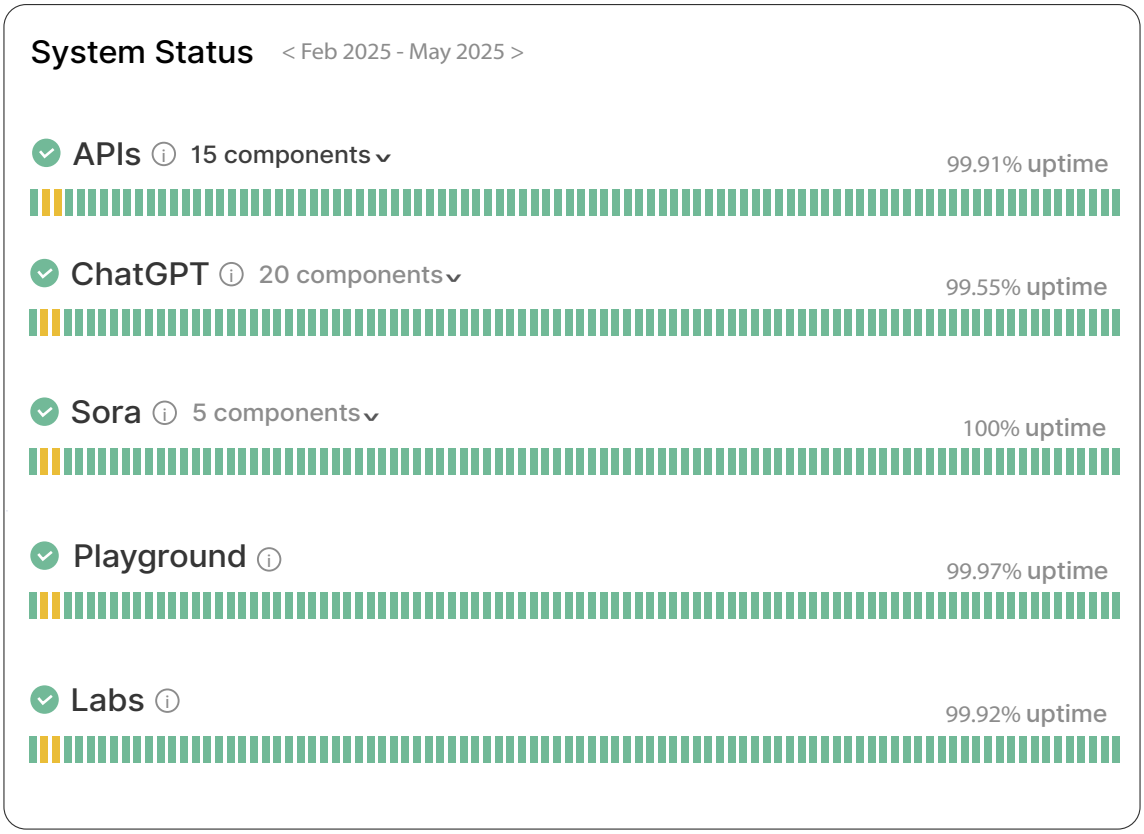
Concepts	OpenAI	Gemini	Claude
Max token	`max_tokens`	`maxOutputTokens`	`max_tokens_to_sample`
Temperature Range	`0.0_2.0`	`0.0_1.0`	`0.0_1.0`
Stop Sequence	`stop` array	X not supported	`stop_sequence`

Implication: Engineering teams must implement custom handling logic per provider, which results in duplicated effort, increased complexity, and limited interoperability.

1.2 Provider Instability and Runtime Failures

Despite high service-level agreements, model APIs from providers such as OpenAI and Anthropic exhibit periodic degradation or downtime due to infrastructure constraints, upstream dependencies, or high demand.

Here's a screenshot of OpenAI's status page from Feb to May 2025.



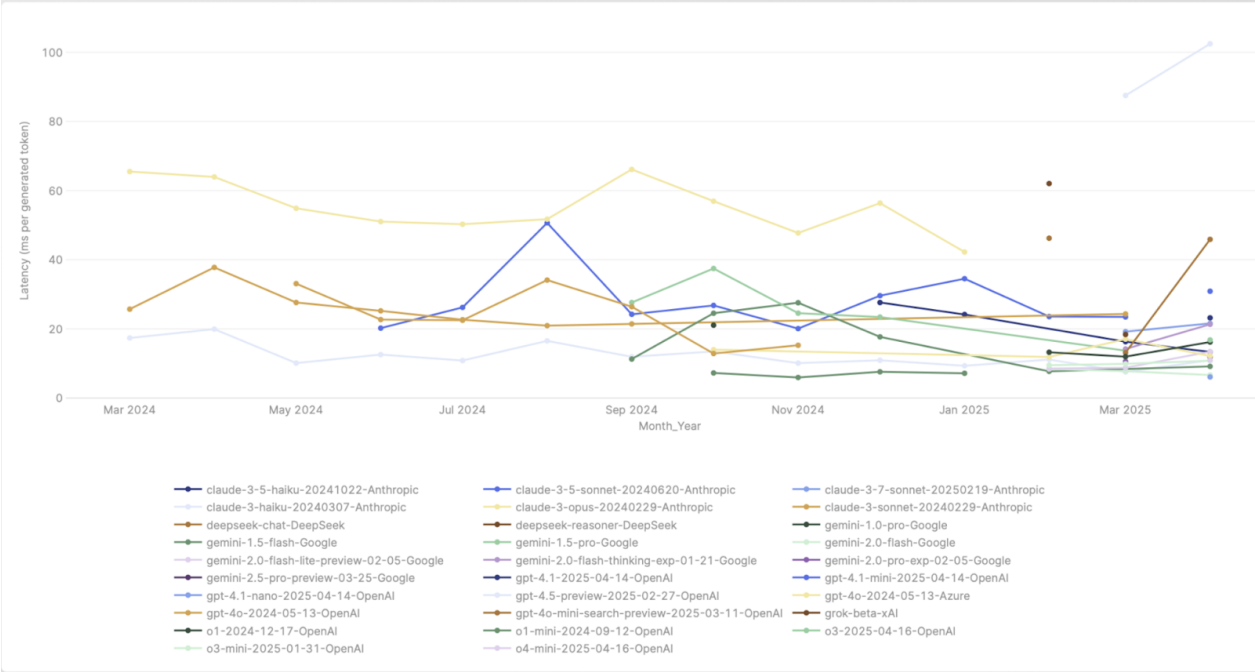
Operational Risk: Applications directly coupled to a single provider are prone to systemic failures during outages or throttling events. Resiliency strategies (e.g., fallbacks, retries) are not standardized across clients.

1.3 Latency Variability and SLA Violations

Model inference latency is neither constant nor guaranteed.
It varies based on:

- Geographical region
- Model version
- Time of day
- Request volume

Here's a graph of the latency variance of a few models over a course of a month



Empirical latency measurements across providers show fluctuations that exceed acceptable SLA
thresholds for production-grade applications.

Consequence: Systems without adaptive routing or latency-aware orchestration risk service degradation and customer dissatisfaction.

1.4 Dynamic Rate Limiting and Throttling Constraints

Model providers enforce rate limits on a per-account, per-model, and sometimes per-tenant basis. These limits can change based on contractual tier, usage history, or platform updates.

Model	Tier	Quota Limit in toker per minute (TMP)	Request per minute
gpt-4o	Enterprise agreement	30M	180K
gpt-4o-mini	Enterprise agreement	50M	300K
gpt-4 (turbo-2024-04-09)	Enterprise agreement	02M	12K
gpt-4o	Default	450K	2.7K
gpt-4o-mini	Default	02M	12K
gpt-4 (turbo-2024-04-09)	Default	450K	2.7K

Engineering Complexity: Without centralized quota enforcement and rate-aware queuing, systems face unpredictable failures or degraded throughput under load.

1.5 Cost Attribution and Budget Governancev

LLM usage incurs significant costs, often billed by:

- Token counts (input + output)
- Model type (base vs. fine-tuned)
- Deployment region or platform (e.g., Azure OpenAI vs. OpenAI direct)

However, billing data is distributed across platforms (e.g., Azure Monitor, AWS CloudWatch, OpenAI Dashboards), making cost attribution across teams, models, or environments cumbersome.

Challenge: Lack of unified cost telemetry impairs budgeting, forecasting, and internal chargeback models.

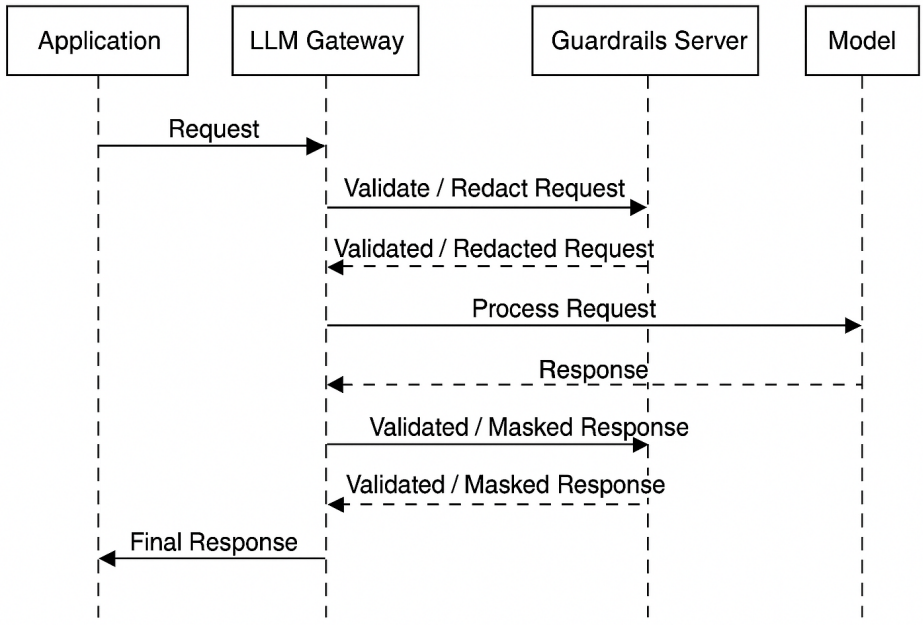
1.6 Redundant Prompt Processing and Semantic Overhead

LLMs frequently process repetitive, low-variance inputs (e.g., greetings, standard queries). A single “Hi” to GPT-4o can consume ~10 tokens. At high volumes, this leads to measurable financial waste.

Optimization Opportunity: Semantic caching — caching responses based on prompt intent rather than exact text — can reduce costs significantly without affecting application fidelity.

1.7 Inconsistent Guardrails and Risk Exposure

Security policies — such as PII redaction, brand tone enforcement, or jailbreak protection — are often inconsistently implemented across applications.



Risk: Without centralized semantic input/output validation, enterprises face elevated exposure to data leaks, regulatory violations, and reputational harm.

1.8 Agentic Applications Face a Fragmented Backend

A single intelligent application may interact with:

- Multiple LLM providers (e.g., OpenAI, Claude, Mistral)
- Enterprise data sources (e.g., SQL, email systems, konwledge bases)
- Authentication and authorization layers
- Operational tools and APIs (e.g., Slack, Jira, Confluence) via MCP Servers

In such a topology, prompt orchestration, session management, and response synthesis become operationally complex—especially when workflows span multiple models and tools.

Architectural Burden: Without an abstraction layer like an **AI Gateway**, each application must independently solve for model selection, routing logic, security enforcement, tool invocation (via MCP), and observability. This creates duplication, brittleness, and scaling friction across teams.

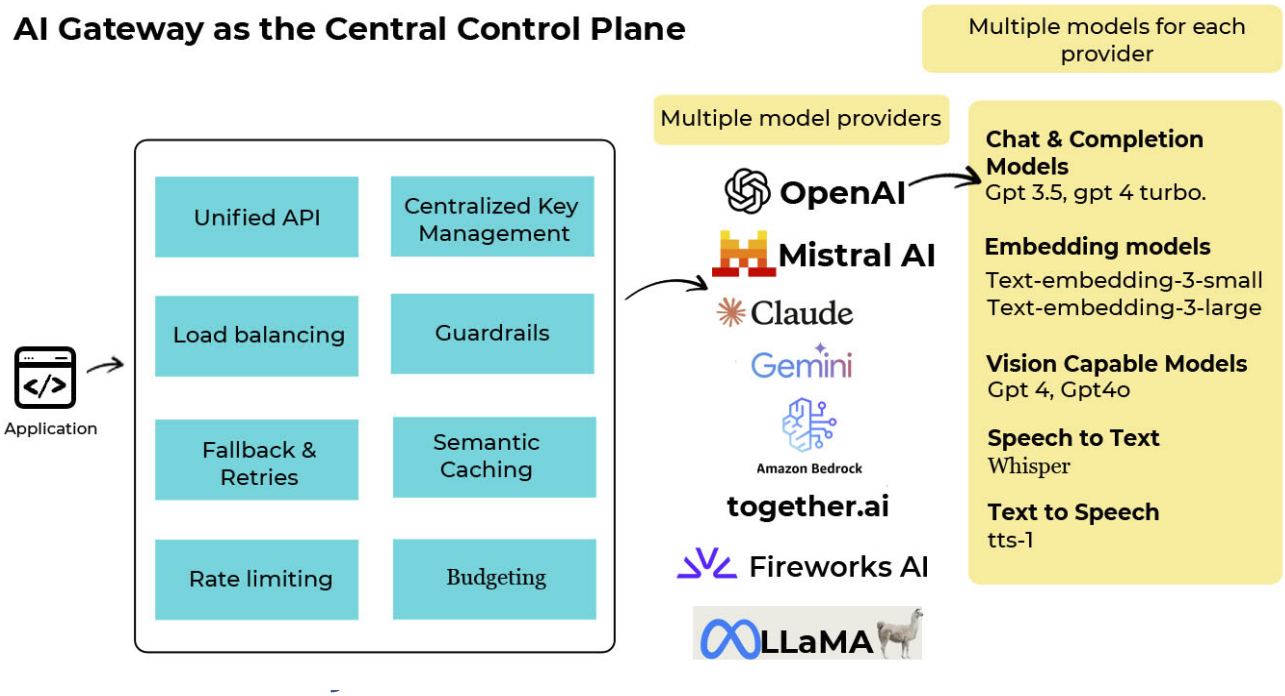
Chapter 2

Introducing the AI Gateway

In response to the architectural challenges outlined in the previous chapter, the AI Gateway emerges as a critical infrastructure layer for enterprise-grade AI adoption. It performs the role of a centralized control plane — abstracting complexity, standardizing access to models, enforcing governance, and optimizing operational efficiency.

What is an AI Gateway?

An AI Gateway is a middleware component that sits between AI-powered applications and various LLM or generative model providers. Its core function is to serve as a unified entry point and control surface for AI workloads, much like how an API Gateway mediates between clients and backend services.



- **Abstraction:** Normalize heterogeneous model APIs into a single interface
- **Routing:** Dynamically select and switch between models and providers
- **Governance:** Enforce policies related to security, compliance, and content safety
- **Optimization:** Implement semantic caching, fallback strategies, and cost controls
- **Observability:** Enable telemetry across prompts, responses, costs, and model behavior

Without an AI Gateway

A product team builds a chatbot that uses GPT-4. Soon, they want to:

- Try Claude for summarization
- Switch to Mistral for EU workloads
- Add fallback if a model times out
- Track usage costs per team
- Enforce guardrails for PII and profanity
- Cache high-frequency prompts

Without an AI Gateway, this requires custom code, manual monitoring, and model-specific hack—leading to brittle infrastructure and duplicated effort.

Key Capabilities of an AI Gateway

Capability	What It Does	Enterprise Example
Unified Model Access	Integrates with 100–250+ LLMs via a single API	One endpoint handles GPT-4, Claude, local Mistral, etc.
Routing & Fallbacks	Routes based on latency, cost, region; retries automatically	. If GPT-4 throttles, automatically switch to Claude for the same prompt.
CLI, SDK, & UI Tools	Access model integration, configs via UI, CLI & SDK	Easy to adopt AI Gateway with minimal friction across interfaces
Rate Limit & Quota Management	Controls request count, token usage, per user/team/model	Limit GPT-4 to 10 requests/min for junior users; throttle high-volume clients
Semantic Caching	Deduplicates prompts at intent level for cost and latency savings	Reuses responses for repeated queries like “Hello” or FAQ requests.
Guardrails & Policy Controls	Enforces content policies, redacts PII, prevents jailbreaks	Automatically strip sensitive data before model submission, enforce intolerance filters.
Multi-Provider Cost Monitoring	Tracks usage & spend across providers, teams, and keys	Provides dashboards for spend per team and model; alert on cost overruns.
Observability & Tracing	Logs full prompt→response path, metadata, user, latency	Trace a user prompt’s path: model A fallback to model B → cost variance → error handling

MCP & Tool Integration	Discover and invoke enterprise tools like Slack, Jira, Datadog	“Send a message to Slack channel” simply becomes gateway logic.
A2A Agent Coordination	Enables structured interaction between multiple agents	Agents use the gateway’s messaging layer for task delegation and context sharing
DeploymentFlexibility - On-Prem/Hybrid	Supports Kubernetes, on-prem, cloud, air-gapped, multi-region deployment	Deploy EU-only models or on-prem for compliance and low latency.
Secret and Key Management	Secure vault for provider credentials	Store API keys securely across teams, projects etc
Advanced Features	Canary testing, batch processing, fine-tuning pipeline integration	Test new model updates safely; batch multiple prompts in one API call for throughput; run fine-tuning jobs.

Chapter 3

Why Traditional API Gateways Fall Short for AI Workloads

While API Gateways excel at managing restful services, they lack the nuanced capabilities required for AI-specific tasks. Here's an in-depth look at the limitations of API Gateways in the context of AI workloads and the specialized functionalities that AI Gateways provide.

Limitations of Traditional API Gateways

Capability	API Gateway	AI Gateway
Traffic Type	Structured, schema-bound REST/GraphQL requests	Natural language, streaming tokens, files, audio, function calls
Semantic Awareness	No content understanding, superficial keyword checks	Deep prompt analysis: redaction, injection detection, bias/ethics filters
Rate Limiting & Token Quotas	Limits requests per API key/IP	Enforces quotas per user/team/model/token count—prevents runaway cost (e.g., infinite loops costing thousands)
Model Feature Normalization	Treats each model as a black box	Uniform interface across heterogeneous model parameters (e.g., stop sequences, token limits)
Fallback & Load Balancing	Not model-aware, only HTTP-level failover	Supports model-level fallback when rate limits hit or latency degrades
Observability	Logs endpoints, HTTP codes, latency	Logs model-specific details: token usage, cost, model selection, prompt—essential for debugging LLM behavior
Security & Compliance	Header-level auth, transport encryption	Content-level guardrails: PII detection, prompt/filtering, hallucination/harm prevention
Multi-Step Orchestration	Limited to single request-response cycles	Integrates with MCP and A2A protocols to orchestrate agents and enterprise tools (e.g., Slack, Jira)
Cost-Aware Routing	No insight into downstream pricing	Routes requests based on token cost, usage budgets, and SLAs
Scalability & Resilience	Built for microservice traffic	Optimized for sub-ms decisions, in-memory enforcement, and fail-safe architecture

Lack of Semantic Understanding

API Gateways are designed to handle structured data and predefined endpoints. They operate effectively with deterministic inputs and outputs, such as JSON payloads. However, AI workloads often involve unstructured data, such as natural language prompts, which require semantic understanding and processing. API Gateways lack the capability to interpret and manipulate such data effectively.

Inadequate Support for Model Specific Parameters

Different LLM providers have varying API structures, parameter names, and configurations. For instance, the parameter for controlling output length might be **max_tokens** in one model and **maxOutputTokens** in another. API Gateways are not inherently designed to handle these discrepancies, leading to increased complexity in managing multiple models.

Observability and Monitoring

API Gateways provide basic monitoring capabilities, such as request counts and response times. However, they do not offer insights into AI-specific metrics like token usage, model performance, or prompt-response analysis. This lack of observability hampers the ability to monitor and optimize AI workloads effectively.

Limited Security and Compliance Features

While API Gateways can enforce authentication and rate limiting, they fall short in addressing AI-specific security concerns. For example, they cannot detect or prevent prompt injection attacks, nor can they ensure compliance with data privacy regulations by redacting sensitive information from AI-generated responses.

Chapter 4

How you should think about build VS buy for AI Gateway for enterprises

To many teams, the idea of building an in-house AI Gateway feels almost obvious at first. After all, it's just routing — a lightweight layer that sits between prompts and models, right? The logic is tempting: write a few wrappers around APIs, add some logging, and move on.

But that illusion of simplicity begins to fracture almost immediately at scale.

What Starts Simple Becomes Operationally Complex

Routing a handful of models is manageable. The engineering effort required to send a prompt to an LLM and receive a response is minimal. But over time, as use cases expand and model options multiply, the complexity curve steepens rapidly.

New models are released weekly. Each has its own interface conventions, parameter tuning, rate limits, and latency profiles. Before long, teams need more than just routing:

- They require fallbacks across models for reliability.
- They need load balancing, usage tracking, and semantic caching.
- They must enforce governance, enable observability, and manage

multi-region deployments.

These are not peripheral features—they are core requirements for deploying AI systems into real production environments. Building them in-house transforms a simple router into a critical, high-maintenance infrastructure system.
And the cost is not just technical debt—it's operational risk. When internal gateways fail or lag behind in functionality, they break customer-facing experiences, slow down product teams, and consume valuable engineering bandwidth.

The Gateway Has Evolved into a Central Control Plane

What was once a mere transport layer is now evolving into the control plane of the AI stack. A modern AI Gateway is expected to:

- **Manage model orchestration** and provider abstraction.
- **Register and route to enterprise tools** via MCP Servers.
- **Mediate communication between agents** through A2A protocols.
- **Provide secure authentication**, rate limiting, and usage-based access.
  Support deployment flexibility across on-prem, cloud, and multi-region environments

Strategic Focus: Build What Different iates, Buy What Scales

The key question for any team should be: Is your AI Gateway part of your product differe ntiation—or part of your foundation?

If your Gateway is not itself a competitive advantage, then building and maintaining it in-house only serves to slow you down. The more time you spend building infrastructure, the less time your team spends delivering value to customers.

In such cases, the logical approach is to adopt an enterprise-grade solution—a platform that's built to handle the fast-evolving AI ecosystem, provide reliability guarantees, and integrate seamlessly into your stack.

Solutions like TrueFoundry's AI Gateway are already trusted by engineering teams looking to move faster, not reinvent. These platforms offer:

- Prebuilt support for the latest models and tools.
- Guaranteed uptime, performance, and compliance.
- Plug-and-play integration with Kubernetes, on-prem deployments, and cloud-native varchitectures.
- Native support for protocols like **MCP and A2A**, powering next-gen automation.

In doing so, they free your team to focus on what matters most: building differentiated AI experiences, not maintaining infrastructure behind the scenes.

Chapter 5

Architecture of AI Gateway

In modern generative AI systems, the AI Gateway functions as the critical proxy layer between applications and language model (LLM) providers. It plays a central role in managing reliability, observability, access control, and cost-efficiency for every request flowing into production.

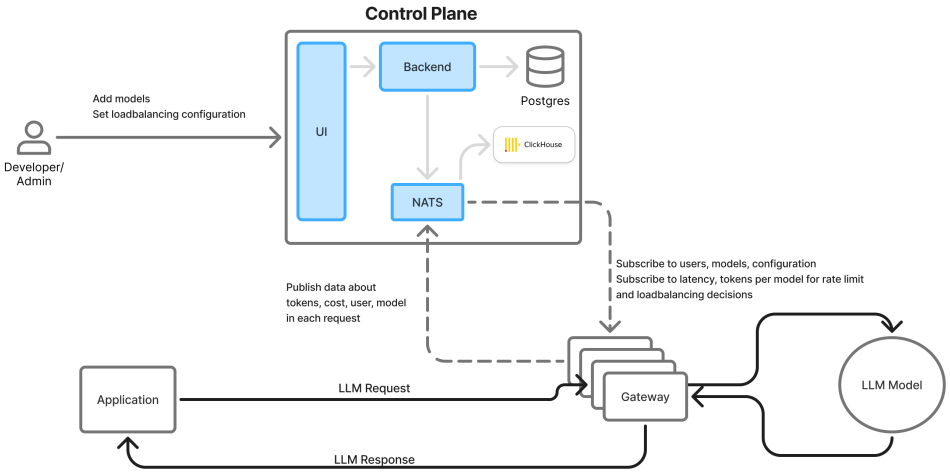
Because the gateway lies in the critical path of production traffic, it must be designed with the following core principles in mind:

Key Architectural Priorities

- **High Availability:** The gateway must not become a single point of failure. Even in the face of dependency issues (like database or queue outages), it should continue serving traffic gracefully.
- **Low Latency:** Since it sits inline with every inference request, the gateway must add minimal overhead to ensure a snappy user experience.
- **High Throughput and Scalability:** The system should scale linearly with load and be able to handle thousands of concurrent requests with efficient resource usage.
- **No External Dependencies in the Hot Path:** Any network-bound or disk-bound operations should be offloaded to asynchronous systems to prevent performance bottlenecks
- **In-Memory Decision Making:** Critical checks like rate limiting, load balancing, authentication, and authorization should all be performed in-memory for maximum speed and reliability.
- **Separation of Control Plane and Proxy Plane:** Configuration changes and system management should be decoupled from live traffic routing, enabling global deployments with regional fault isolation.

TrueFoundry's AI Gateway Architecture

TrueFoundry's AI Gateway embodies all of the above design principles, purpose-built for low latency, high reliability, and seamless scalability



Key Architectural Priorities

- **Built on Hono Framework:** The gateway leverages Hono, a minimalistic, ultra-fast framework optimized for edge environments. This ensures minimal runtime overhead and extremely fast request handling.
- **Zero External Calls on Request Path:** Once a request hits the gateway, it does not trigger any external calls (unless semantic caching is enabled). All operational logic is handled internally, reducing risk and boosting reliability.
- **In-Memory Enforcement:** All authentication, authorization, rate-limiting, and load-balancing decisions are made using in-memory configurations, ensuring sub-millisecond response times.
- **Asynchronous Logging:** Logs and request metrics are pushed to a message queue asynchronously, ensuring that data observability does not block or slow down the request path.
- **Fail-Safe Behavior:** Even if the external logging queue is down, the gateway will not fail any requests. This guarantees uptime and resilience under partial system failures.
- **Horizontally Scalable:** The gateway is CPU-bound and stateless, which makes it easy to scale out. It performs efficiently under high concurrency and low memory usage.

Control Plane & Data Flow

TrueFoundry separates the control plane (management) from the data plane (real-time traffic routing) for scalability and flexibility.

Components Overview of the AI Gateway:

- **UI:** Web interface with an LLM playground, monitoring dashboards, and config panels for models, teams, rate limits, etc.
- **Postgres DB:** Stores persistent configuration data (users, teams, keys, models, virtual accounts, etc.)
- **ClickHouse:** High-performance columnar database used for storing logs, metrics, and usage analytics.
- **NATS Queue:** Acts as a real-time sync bus between control plane and distributed gateway pods. All config/state updates are pushed through NATS and instantly available in all regions.
- **Backend Service:** Orchestrates config syncing, database updates, and analytics ingestion.
- **Gateway Pods:** Stateless, in-region, lightweight proxies that handle actual LLM traffic. They consume NATS messages and perform all logic in-memory, with no external dependencies.

Performance Benchmarks for TrueFoundry's AI Gateway

TrueFoundry's Gateway has been thoroughly benchmarked for performance under vproduction-like loads:

- **250 RPS on 1 CPU/1GB RAM** with only **3 ms added latency.**
- **Scales efficiently up to 350 RPS** per pod before hitting CPU saturation, beyond which you can add replicas.
- Supports **tens of thousands of RPS** with horizontal scaling across regions.
- **No additional latency** even with multiple rate-limit, auth, and load-balance rules in place.

Why This Matters

If you're running genAI workloads at scale, or planning to integrate multiple LLMs (OpenAI, Claude, open source, etc.), the gateway becomes the foundation of your stack.

TrueFoundry's design ensures:

- You can route and scale safely across providers.
- Apply fine-grained controls at user/team-level.
- Maintain observability and governance across the system.
- Do all of this without impacting latency or reliability.

Chapter 6

Future of AI Gateway - Integration with MCP Server and A2A protocol

An MCP Server abstracts enterprise APIs and services into a standard, LLM-consumable interface. It allows AI agents to discover, authenticate, and invoke enterprise functionality through consistent, machine-interpretable schemas.

What Is an MCP Server?

At its core, an MCP Server transforms internal or third-party APIs into structured service descriptions that can be discovered, authenticated, and invoked by LLMs and agents.

Each MCP Server presents functionality in a machine-readable, context-aware schema, enabling AI agents to reason about what the system can do and how to interact with it.

Agents use MCP Servers to:

- Discover available tools and operations, dynamically.
- Authenticate seamlessly via enterprise identity providers such as Okta or Azure AD.
- Invoke those operations through natural language tasks translated into structured RPC calls.

For example, a Slack MCP Server abstracts capabilities like listing channels, retrieving messages, and posting replies—making them fully interpretable and executable by an LLM agent without explicit hardcoding. Rather than engineering against Slack's REST API directly, agents work through a consistent semantic interface.

AI Gateway + MCP: Unified Execution Layer

With MCP integration, the AI Gateway becomes far more than a router for LLMs. It transforms into a federated control plane that unifies access across all AI models, enterprise tools, and internal systems. Through a single gateway endpoint, users and agents can orchestrate end-to-end workflows involving:

AI models, such as GPT-4, Claude, and fine-tuned domain models.
Enterprise tools, including Slack, Jira, Confluence, and others.
Internal services, from proprietary APIs to data platforms and observability stacks.

Consider a user command like: "Create Jira tickets from urgent Slack messages in #support." With MCP, this task is entirely agent-executable. The agent queries the Slack MCP to gather relevant messages, summarizes the content, then invokes the Jira MCP to generate structured tickets—without requiring manual integration, scripting, or user intervention.

Core Capabilities of the MCP-Enabled Gateway

To support this functionality at scale, the Gateway + MCP architecture incorporates several key primitives:

Registry and Discovery

MCP introduces a central MCP registry within the gateway. All MCP servers according to right access control become instantly discoverable based on permissions and organizational policies. This registry supports:

This dramatically reduces the integration overhead for new systems. If a service is MCP-compliant, it can be discovered and used within minutes.

Out-of-the-Box MCP Server Support

To accelerate enterprise onboarding, TrueFoundry and the broader ecosystem offer **prebuilt MCP Servers** for popular tools such as:

- Slack
- Confluence
- Sentry
- Datadog

These integrations are code-free—ready to deploy and use in workflows immediately—enabling organizations to move from prototype to production in record time.

Bring Your Own MCP Server

For internal APIs or proprietary services, organizations can build and deploy their own MCP Servers using open SDKs. Once registered with the Gateway, these services are fully integrated into the discovery, routing, and observability stack. Any service that speaks MCP can be orchestrated by agents as part of their workflows.

Enterprise-Grade Authentication and Authorization

MCP is designed with the security and compliance requirements of large organizations in mind. It integrates seamlessly into existing identity and access control systems:

- Federated Authentication ensures SSO-based login using providers like Okta, Azure AD, or any standard IdP—eliminating the need for new credentials or siloed auth stacks.
- Role-Based Access Control (RBAC) policies govern which users or agents can access specific MCP Servers, tool functions, or datasets.
- OAuth 2.0 and Dynamic Discovery allow secure session management with scalable permission enforcement, all mediated transparently through the Gateway.

Agentic Task Execution

Perhaps the most transformative impact of MCP is that it empowers autonomous agents to execute complex, cross-system workflows using only natural language as input. This includes:

- Multi-tool task automation: Orchestrating actions across Slack, Jira, Datadog, etc.
- Dynamic code generation: Where agents synthesize and run executable code against
- MCP tools—transforming how operators, analysts, and developers interact with their
- environments.

Through MCP, the Gateway becomes an intelligent execution substrate—bridging LLMs and operational systems at runtime.

Built-In Observability, Auditing, and Governance

For internal APIs or proprietary services, organizations can build and deploy their own MCP Servers using open SDKs. Once registered with the Gateway, these services are fully integrated into the discovery, routing, and observability stack. Any service that speaks MCP can be orchestrated by agents as part of their workflows.

To operate securely at scale, enterprises must track every interaction across their AI stack. The Gateway + MCP infrastructure includes first-class observability features:

- End-to-end tracing for user, agent, and tool interactions across all systems
- Structured metadata tagging, enabling analysis by team, environment, tool, or purpose
- Usage and cost analytics to support budgeting, chargebacks, and policy enforcement
- This telemetry supports full regulatory compliance, operational debugging, and strategic optimization.

The Rise of Multi-Agent Systems

Modern enterprises no longer build monolithic AI applications. Instead, they are decomposing logic across multiple intelligent agents, each optimized for a domain-specific capability. For example:

- **A Planner Agent** decomposes a user task into subtasks.
- **A Retriever Agent** gathers relevant documents or data.
- **An Executor Agent** performs API calls or tool interactions via MCP.
- **A Summarizer Agent** compiles output into human-readable results.

These agents must collaborate dynamically, passing context, invoking each other's capabilities, and aligning around shared goals. Without a common protocol, such coordination becomes brittle and ad hoc.

What Is the A2A Protocol?

These agents must collaborate dynamically, passing context, invoking each other's capabilities, and aligning around shared goals. Without a common protocol, such coordination becomes brittle and ad hoc.

- Discover each other's capabilities, interfaces, and trust levels.
- Exchange messages and payloads with contextual state.
- Invoke functions exposed by other agents.
- Negotiate task delegation and execution pathways.
- Log interactions for traceability, observability, and compliance.

This protocol ensures that agents are not just coexisting, but collaborating in a governed and enterprise-safe manner.

Role of the AI Gateway in A2A Communication

The AI Gateway plays a central role in A2A communication by acting as a semantic message bus and security policy layer between agents. It enforces communication standards, authenticates messages, tracks conversation context, and logs all interactions for downstream analytics and compliance.

Core Functions of the Gateway in A2A:

- **Secure Message Passing:** Encrypt, sign, and route messages across agent boundaries
- **Context Propagation:** Maintain shared conversation state, execution history, and agent memory.
- **Access and Capability Control:** Allow or restrict inter-agent calls based on role, environment, or trust level.
- **Full Traceability:** Log every A2A interaction as a first-class telemetry event.

In essence, the AI Gateway serves as the control plane and router for all intelligent agent traffic in the enterprise.

Governance and Observability for A2A

A2A workflows introduce new operational risks. Rogue agents, infinite loops, or permission overreach could severely impact system integrity. The AI Gateway mitigates these risks by embedding fine-grained control and visibility:

- **RBAC for Agent Roles:** Limit which agents can call others or invoke specific MCP actions.
- **Telemetry Hooks:** Log inter-agent events, including message content (redacted), timestamps, outcomes, and cost.
- **Policy Enforcement:** Use dynamic policies to control message types, data propagation, and execution conditions.
- **Replay and Debugging:** Reconstruct agent interactions for compliance audits or fault diagnosis.

Chapter 7

Making AI Gateways Enterprise-Ready - From Self-Hosted Models to and Multi-Region Gateways

As organizations transition from experimentation to production-scale AI systems, deployment flexibility and operational resilience become critical. Whether deploying across global data centers, meeting strict compliance needs, or managing diverse model types, enterprises require an AI Gateway that's not just cloud-compatible—but cloud-agnostic, region-aware, and Kubernetes-native.tt

Self-Hosting LLMs, SLMs, and Fine-Tuned Models on Kubernetes

Enterprises increasingly seek control over model hosting—either to meet regulatory constraints, reduce latency, or fine-tune models with proprietary data. The AI Gateway must support direct integration with self-hosted models running on Kubernetes clusters or secure infrastructure environments.

Key capabilities include:

• **Native Support for Self hosting LLMs:** Seamlessly connect self-hosted LLMs (e.g., LLaMA, Mistral, Falcon) as model backends within the Gateway.

• **Fine-Tuned Model Routing:** Register organization-specific fine-tunes as first-class citizens in the Gateway and route traffic seamlessly.

• **Multi-Model Server Support:** Out-of-the-box compatibility with popular open-source model servers like vLLM, SGLang, Triton Inference Server, Text-Generation-Inference, and Ray Serve allows teams to host both base and fine-tuned models with minimal custom integration.

• **Containerized Deployment:** Run any model—LLMs, SLMs, or task-specific fine-tunes—in a containerized environment using Helm charts, Docker containers.

• **GPU Scheduling & Auto-Scaling:** Provision and manage GPU resources using Kubernetes-native autoscaling strategies and schedulers. Scale replicas up or down based on load, latency, or budget constraints—optimizing both performance and cost.

• **Gateway Integration:** Once deployed, these models are registered with the AI Gateway as accessible endpoints—complete with observability, rate limits, caching, and guardrail enforcement. This ensures production-grade routing, monitoring, and policy control even for internal workloads.

This allows AI platforms to operate with full flexibility—mixing public APIs and internal model endpoints under a single unified control plane.

Hybrid and On-Prem Deployments for Gateway

Not every workload can be served from the cloud. Enterprises operating under strict compliance (e.g., financial institutions, defense, healthcare) often require on-premise or hybrid setups where sensitive data and compute must remain inside organizational firewalls.

The AI Gateway supports:

• **Secure On-Prem Deployments:** Run the Gateway itself inside secure environments—on bare metal, Kubernetes clusters, or private VPCs—while retaining full orchestration and governance features.

• **Air-Gapped Mode:** Operate in offline environments where no external network access is permitted; all models, logs, and access controls stay local.

• **Unified RBAC and Observability:** Apply consistent access control and traceability rules, regardless of whether traffic is handled in the cloud or on-prem.

This hybrid flexibility ensures compliance without sacrificing innovation or speed.

Multi-Region Availability and Global Failover

Modern enterprises are distributed across continents. Serving users globally demands low-latency model access, regional compliance, and resilient failover.

To support this, the Gateway enables:

• **Geo-Redundant Gateway Deployments:** Run the Gateway in multiple regions—US, EU, APAC—with region-specific model registries and resource pools.

• **Region-Aware Routing:** Automatically direct traffic to the lowest-latency or policy-compliant model endpoint based on user location or request origin.

• **Failover Strategies:** Gracefully degrade to alternate regions or providers in the event of a local failure, using fallback rules and retries built into the Gateway.

This architecture supports global scalability with local guarantees—an essential foundation for AI-native platforms operating across borders.

Chapter 8

Security, Compliance, and Governance in AI Gateways

This chapter outlines the key controls and enforcement mechanisms required to deploy LLM systems responsibly and securely at scale.

Federated Authentication and RBAC

Enterprises rely on centralized identity management to enforce secure access.

The AI Gateway must integrate natively with enterprise IdPs (Identity Providers) like Okta, Azure AD, Google Workspace, or any OIDC-compliant SSO provider.

Key capabilities:

• SSO Integration: Support for SAML, OIDC, or OAuth2-based login flows with session management.

• Granular Role-Based Access Control (RBAC): Enforce per-user, per-team, or per-agent access to models and workspaces for model deployment.

Prompt and Output Validation (Guardrails)

AI systems generate content dynamically—which introduces unique risks like prompt injections, hallucinations, or policy violations. AI Gateways act as the semantic firewall, enforcing content-level governance.

Examples of semantic guardrails:

• **PII Detection and Redaction:** Automatically scrub names, phone numbers, credit card info, or government IDs before passing prompts to LLMs.

• **Profanity and Toxicity Filters:** Intercept unsafe or offensive language using NLP classifiers or regex policies.

• **Prompt Injection Protection:** Detect injection attacks attempting to override system instructions or jailbreak policies.

• **Output Policy Enforcement:** Apply tone guidelines, brand compliance rules, or custom content classifiers before final response delivery.

Secure Key and Credential Management

AI systems often need to invoke external APIs or tools (via MCP Servers), requiring access to sensitive credentials. Gateways must ensure these secrets are stored, used, and rotated securely.

Best practices supported by the Gateway:

• **Centralized Secret Vaults:** Use services like AWS Secrets Manager, HashiCorp Vault, or built-in encrypted stores to manage keys and tokens.

• **Scoped Credentials:** Ensure API keys or OAuth tokens are scoped to the least-privileged task and not shared across workflows.

• **Token Rotation:** Automate credential expiry and renewal to reduce long-term exposure.

Observability, Auditing, and Traceability

Security without traceability is incomplete. Every model interaction, tool invocation, or agent handoff must be observable and auditable.

Observability capabilities include:

• **Request Tracing:** Track the full lifecycle of every request—prompt, model selection, fallback path, cost, and output.

• **Metadata Logging:** Log attributes like user ID, team, tool, region, latency, and tokens consumed.

• **Audit Trails:** Immutable logs of all access and usage events, supporting forensic analysis and compliance audits.

• **Cost Attribution:** Break down spend by team, user, tool, or model to support budgeting and internal chargebacks.

Data Residency and Compliance Readiness

Enterprise customers across regions must meet jurisdiction-specific data requirements (e.g., GDPR, HIPAA, SOC2, FedRAMP).

Enterprise-ready features include:

• **Regional Model Routing:** Route prompts only to models hosted in compliant jurisdictions (e.g., EU for GDPR).

• **Data Encryption:** Encrypt data in transit (TLS 1.2/1.3) and at rest using enterprise-grade encryption standards.

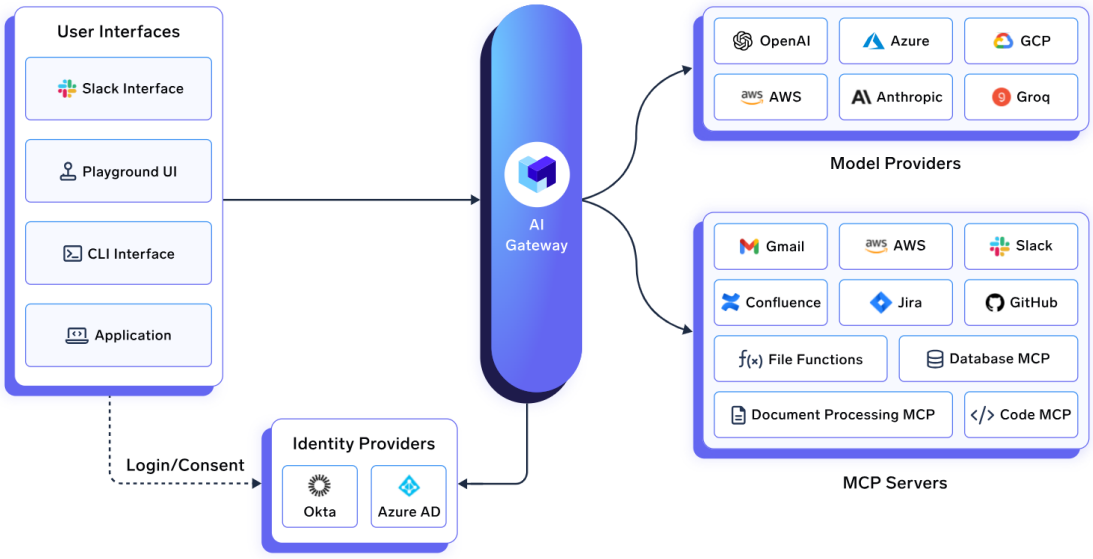
• **Compliance Certifications:** Ensure the gateway infrastructure and hosting platform meet industry certifications (ISO 27001, SOC2, HIPAA, etc.).

• **Data Retention Controls:** Configure TTLs (Time-to-Live) for prompts, responses, logs, or personally identifiable data.

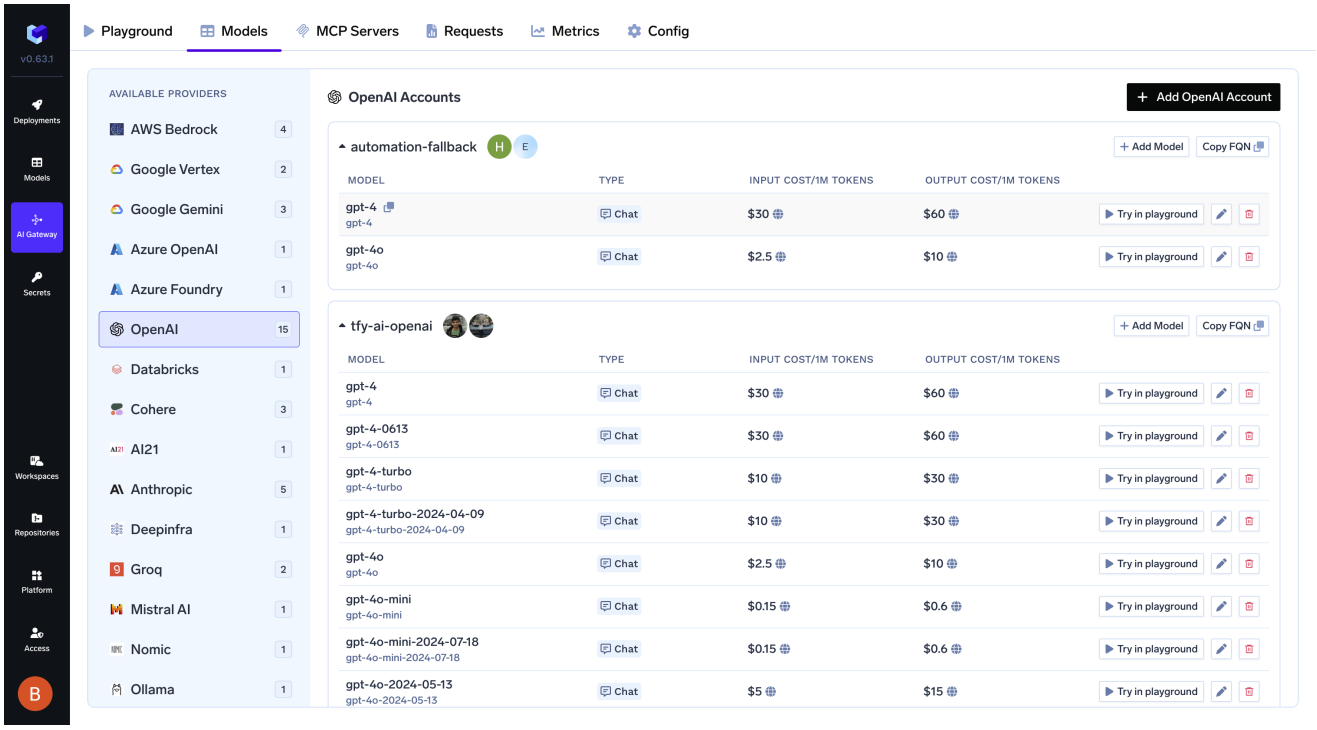
Chapter 9

Inside TrueFoundry's AI Gateway - The Operational Layer for Agentic AI

As enterprises scale their GenAI workloads across teams, tools, and models, they need a production-ready gateway that offers not just abstraction—but operational control, developer ease, and enterprise-grade security. This chapter provides a walkthrough of the TrueFoundry AI Gateway platform, covering how it brings model routing, tool orchestration, observability, and governance into one unified interface.



Unified Model Management



TrueFoundry's Gateway unifies access to both cloud based and self-hosted LLMs via a single, standardized interface. The Models tab displays all available models—whether served by OpenAI, Groq, Claude, or self-hosted on Kubernetes using vLLM, SGLang, or Triton.

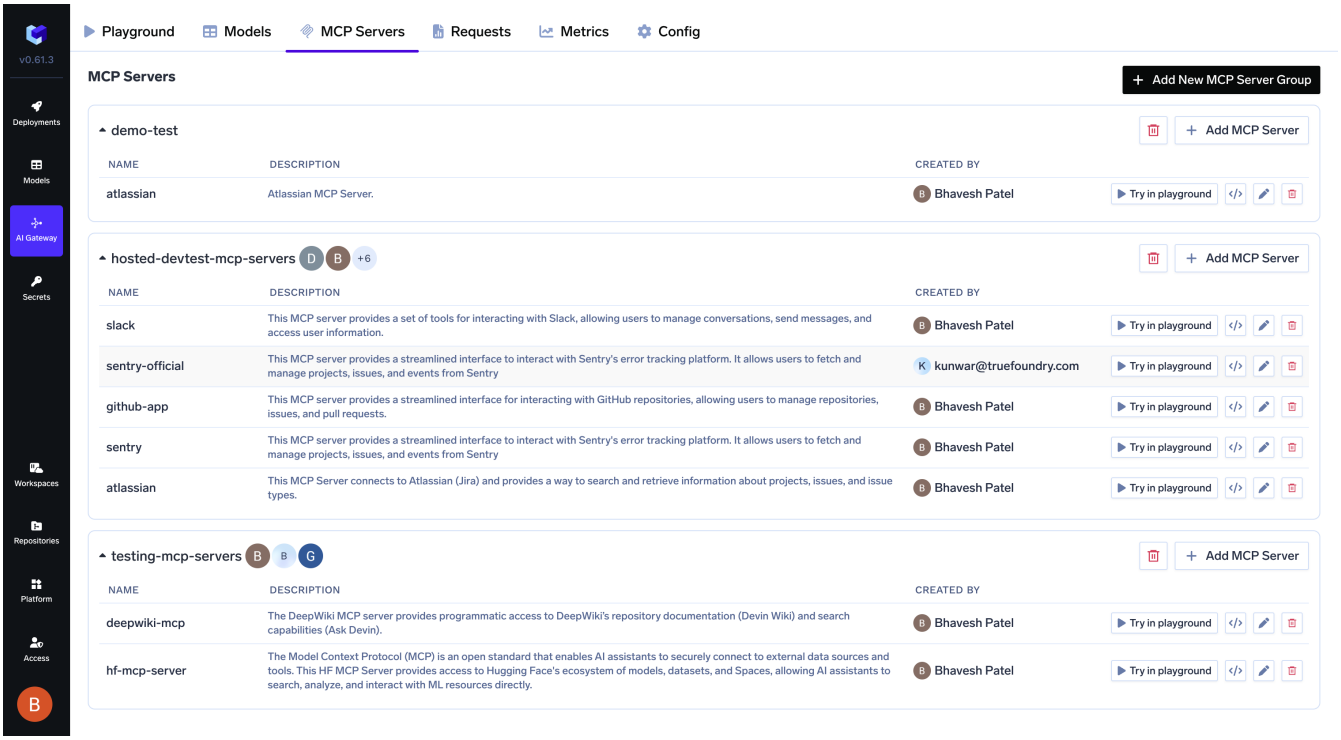
Each model entry shows token costs, input/output formats, and routing configurations. Engineers can instantly add or remove models and assign them to specific accounts or environments. Self-hosted models—including fine-tuned variants—can be deployed via Helm charts or Docker and instantly exposed through the Gateway with no SDK changes.

The Gateway supports:

- Multi-provider access (e.g., OpenAI, Azure, Groq)
- Deployment of self hosted models (e.g., LLaMA, Mistral) on Kubernetes

This abstraction drastically simplifies integration complexity and gives teams flexibility to optimize across dimensions like latency, control, and token economics.

MCP Server Integration



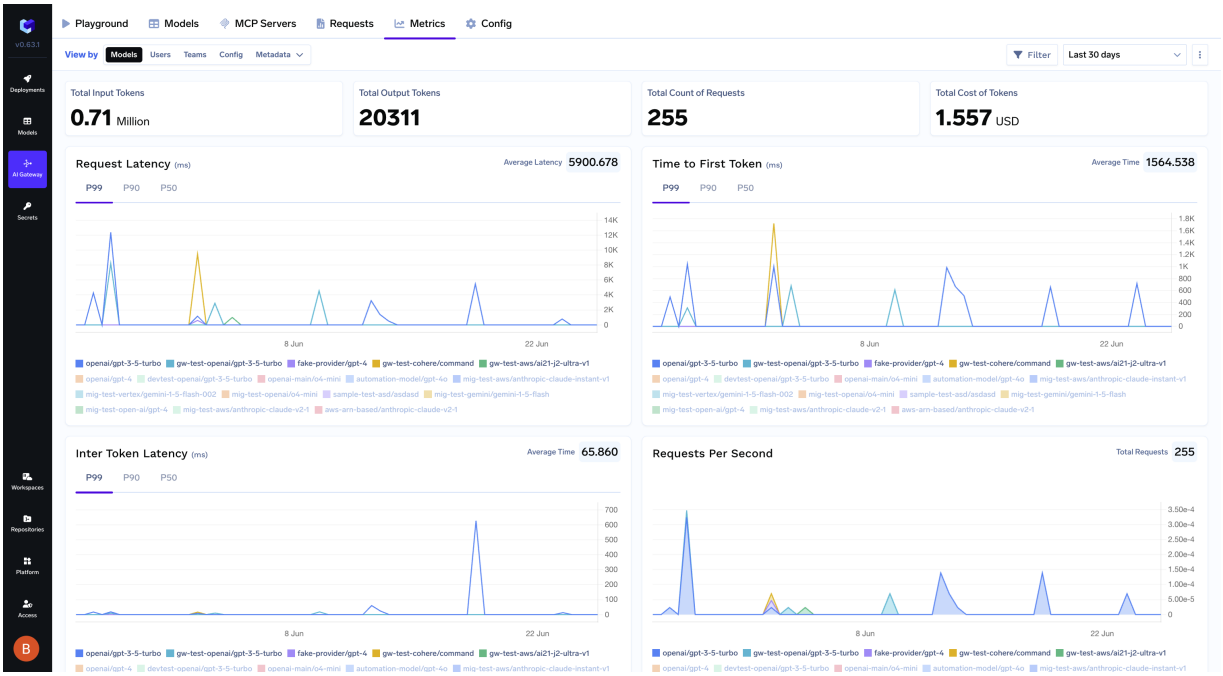
Beyond model access, the Gateway introduces MCP Servers as the standardized layer to connect with external tools and APIs. From Slack to Jira, GitHub to Sentry, MCP Servers act as pluggable, schema-defined interfaces that agents or LLMs can invoke directly within prompts.

Each MCP Server is:

- Fully discoverable via the Gateway interface or API
- Governed by access control rules (RBAC, auth tokens)
- Callable from any model or agent execution

You can register internal tools as MCP Servers using a simple API spec, turning private APIs into agent-compatible functions. In the UI, tools are grouped, previewable in the playground, and testable instantly—dramatically reducing the need for middleware or hardcoded logic.

Built-In Observability and Cost Analytics



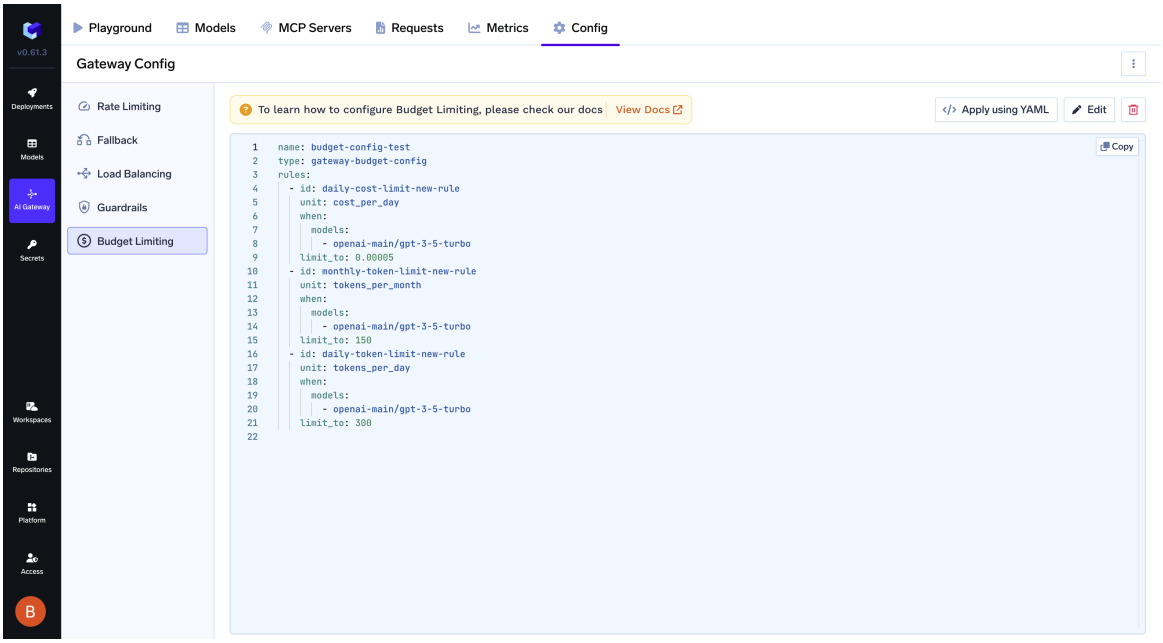
The Metrics tab provides deep visibility into model performance across time, teams, and metadata. It includes request volume, latency trends, inter-token delay, and token-level cost attribution across teams, models, and users.

You can:

- Track performance metrics like P99 latency or RPS per model
- Analyze cost patterns across environments or regions
- Filter telemetry based on request metadata

This makes TrueFoundry's Gateway not only a routing layer—but also a governance system that helps platform teams manage budgets, detect failures, and optimize usage.

Policy Enforcement & Budget Governance



Using the Gateway Config tab, admins can define access policies, fallback rules, and budget limits using YAML-based configuration.

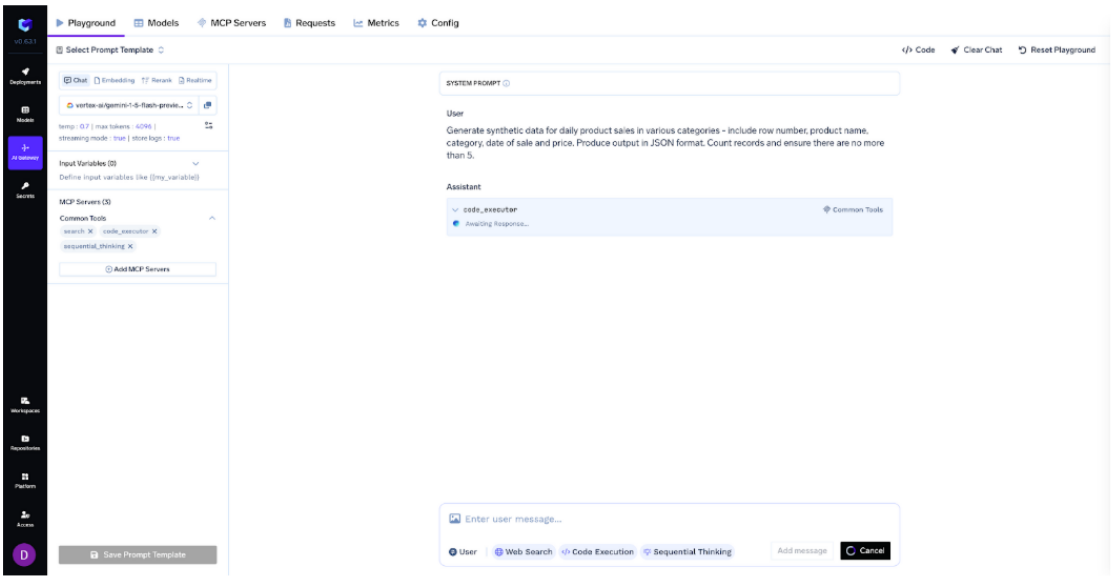
- Rate limiting
- Fallback
- Load balancing
- Budget Limiting

For example:

yaml
CopyEdit
```
- id: daily-cost-limit
  unit: cost_per_day
  models:
    - openai-main/gpt-3.5-turbo
  limit_to: 500.0
```

You can also define token quotas per team, limit access to specific models, or configure fallback routing (e.g., use Claude if OpenAI is overloaded). These controls help organizations ensure predictable usage and reduce overages—even as LLM usage scales.

Developer Playground for Prompting & Code Generation



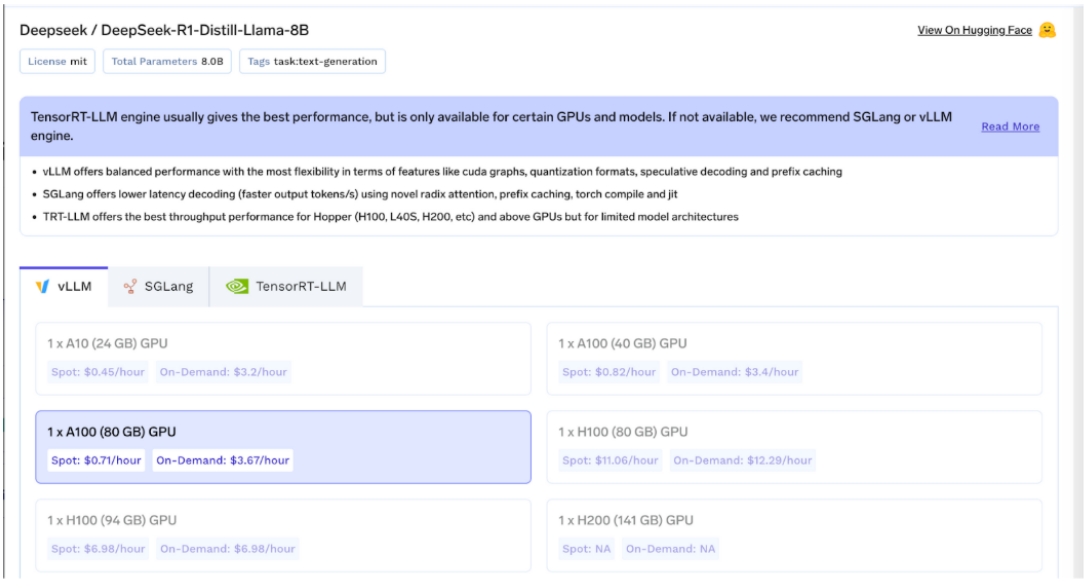
The Playground gives developers a zero-friction environment to prototype prompts and test agent workflows. You can bind MCP Servers to prompts, simulate tool execution, and preview agent behavior across providers.

For every configuration:

- It generates production-ready Python or Curl snippets
- It logs tool interactions and model output side-by-side
- It supports prompt variables, streaming responses, and retry logi

This serves both experimentation and handoff to engineering—making it faster to go from prompt design to deployable code.

Advanced Deployment Flexibility



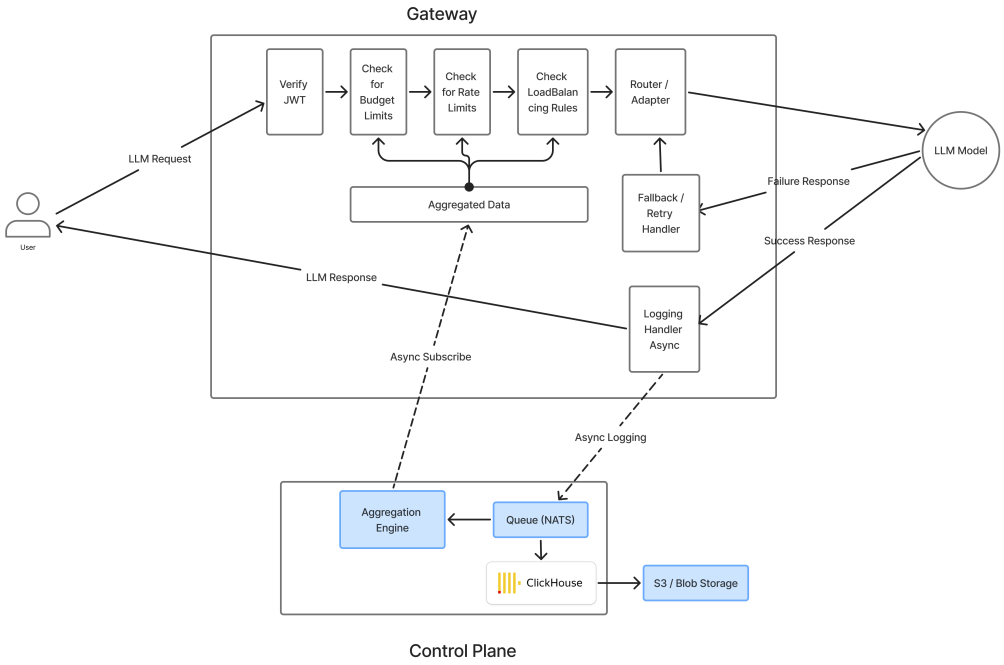
TrueFoundry is Kubernetes-native. Teams can self-host models or the entire Gateway stack within their private infrastructure or on the cloud of their choice. The platform integrates with:

- Model servers like vLLM, Triton, SGLang, KServe, TorchServe
- GPU orchestration with horizontal scaling, affinity rules
- Observability backends like Prometheus, Datadog, or OpenTelemetry

Containerized model deployment, federation across cloud and on-prem, and regional failover are all supported—making the Gateway truly production-ready for enterprise-grade workloads.

Sub-3ms Latency and Scalable Ar-

Here's how a LLM request flows through the gateway:



TrueFoundry's Gateway is designed to be extremely lightweight and performant—even under enterprise-scale workloads. At runtime, every request is handled in-process with no hot path dependency on external databases or services. This architectural decision ensures predictable, ultra-low latency even under high concurrency.

In benchmarks:

- Each Gateway pod adds <3ms overhead to LLM requests at the 99th percentile
- A single pod can handle 250+ RPS on 1 CPU and 1 GB RAM
- Horizontal scaling across regions allows tens of thousands of RPS, with full observability and consistency

This level of performance is critical for use cases such as streaming chat UIs, real-time agent coordination, or latency-sensitive workflows integrated into operational systems.

Standalone & Cloud-Edge Deployment Options

TrueFoundry supports flexible deployment models tailored to enterprise needs and infrastructure diversity

- SaaS–Hosted Gateway, available globally with full multi-region support.
- Self-Hosted Gateway on Kubernetes, within VPCs, on-premise clusters, or air-gapped environments—ideal for compliance-sensitive industries.
- Edge Placement options to run the Gateway near model inference endpoints for minimal latency.

Why this matters:

Whether you're operating entirely in public cloud, deploying inside private data centers, or bridging both via hybrid topology, the Gateway adapts to your architecture. It ensures consistent enforcement of policies, unified observability, and low-latency performance across deployment environments.

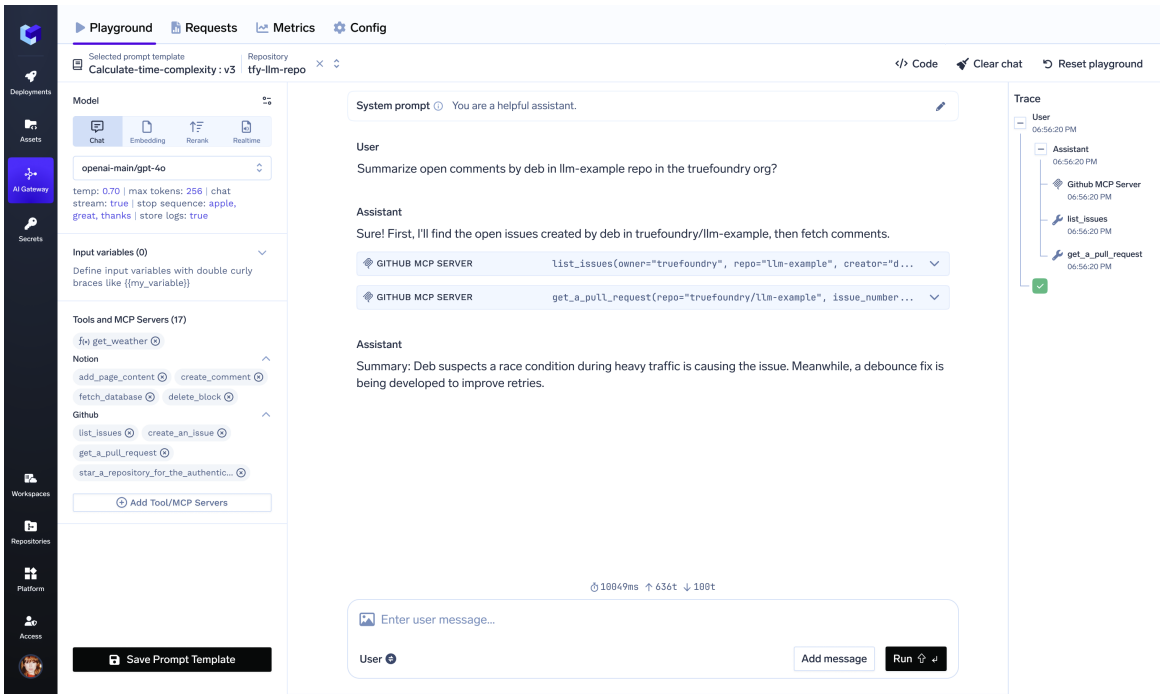
Whether you're operating entirely in public cloud, deploying inside private data centers, or bridging both via hybrid topology, the Gateway adapts to your architecture. It ensures consistent enforcement of policies, unified observability, and low-latency performance across deploy-ment environments.

Conclusion

The rise of agentic systems has reshaped how enterprises approach automation, knowledge workflows, and human-machine interaction. But deploying these systems into production environments is far from trivial. What starts as a model integration challenge quickly spirals into an orchestration, governance, cost-control, and security problem. In this landscape, the AI Gateway is not a luxury—it is foundational infrastructure.

This book has explored how the AI Gateway fills the massive operational and architectural gaps left by traditional API gateways. It unifies access to models, enforces semantic guardrails, provides advanced observability, routes intelligently based on performance or cost, and abstracts the ever-evolving LLM ecosystem into a consistent, compliant, and production-ready interface. Most critically, it evolves into the control plane for AI-powered enterprises, integrating not only with models but also with enterprise tools through MCP Servers and coordinating agent-based workflows via A2A protocols.

For any company building AI-first products or integrating AI across business units, the Gateway becomes the fabric that ensures scalability, security, and speed. And while some may be tempted to build such a system in-house, the cost of maintenance, the pace of innovation, and the infrastructure burden make it a risky long-term strategy.



TrueFoundry's AI Gateway offers a hardened, enterprise-grade platform designed to meet the demands of modern AI deployments. With built-in support for multi-provider orchestration, advanced security controls, observability, semantic caching, and seamless MCP + A2A integration, it empowers teams to focus on building AI-native capabilities—without reinventing the plumbing beneath.