

Request for Proposal: Enterprise LLM Gateway and AI Platform

Template from RFPhub.com. Replace everything in square brackets, adjust the Must-have/Nice-to-have priorities, and delete this note and the instructions section before you issue the RFP.

Instructions for the buyer (delete before issuing)

An LLM gateway sits in the path of every model call your applications make, so the evaluation involves platform engineering, application teams, security, privacy and finance at once, and most of the work is agreeing on scope before vendors get involved. The example schedule below runs about 12 weeks from kickoff to contract award; stretch it if you need a self-hosted deployment in several regions or a formal privacy review of prompt logging.

1. **Inventory current LLM use and agree on scope (weeks 1-2).** List every application that calls a model today, which providers and models it uses, where its API keys are stored and who pays the bill. Include self-hosted models. Record the numbers vendors will price against: applications, teams, monthly requests and tokens, current spend and the regions you operate in.
2. **Assemble the buying team.** Include the platform or AI engineering team that will run the gateway, two or three application teams that will use it, security, privacy and legal, AI governance, network, finance or FinOps, and procurement. Bring privacy in early: the gateway can log every prompt and response, so decide up front what may be logged, what must be redacted, how long logs are kept and where they are stored.
3. **Settle the deployment model, and decide whether you need an RFI (weeks 2-4).** Decide whether prompts may pass through a vendor-hosted service, or whether the gateway must run in your own cloud account or data center. Decide too whether you need a gateway only, or a broader AI platform with prompt management, evaluation and agent tooling. If either question is still open, if you are weighing an open-source gateway against a commercial one, or if your long list has more than five vendors, issue a short RFI first and use the answers to build a shortlist.

4. **Tailor and issue the RFP (weeks 4-7).** Delete requirements that don't apply, adjust priorities, and add your environment details to Section 1. Prepare a test set now for the proof of concept: representative prompts, prompts containing masked personal data, and prompt-injection attempts. Give vendors three weeks to respond, with a written question-and-answer window in the first week.
5. **Score the responses (weeks 7-8).** Have each evaluator score independently against the rubric before comparing notes, and set aside any vendor that misses a Must-have. Check every 'yes' for whether the feature is generally available today, in preview or on the roadmap, and whether it needs a paid edition or an add-on.
6. **Run a proof of concept with two or three vendors (weeks 8-11).** Route real traffic from one or two representative applications through each gateway. Force a provider outage, hit a hard budget limit, measure the latency the gateway adds with guardrails on and off, run your test set through the guardrails, and check that logs are redacted as configured. Reconcile the gateway's cost report with a provider invoice, and call references of a similar size.
7. **Negotiate and award (weeks 11-12).** Use the pricing table to compare like for like, keeping gateway fees separate from model usage. Settle whether model usage is billed by the vendor or directly by providers, renewal caps, overage terms and exit assistance before you sign, not after.

Contents

1. Introduction and Background
2. Project Objectives
3. Scope of Work
4. Requirements
5. Vendor Questions
6. Evaluation Criteria and Scoring
7. Pricing Response
8. Submission Guidelines
9. Timeline

1. Introduction and Background

[Company Name] is seeking proposals for an enterprise LLM gateway that gives our applications governed access to large language models from multiple providers, and to our own self-hosted models, through one API, with routing, cost controls, logging and guardrails. This RFP describes our requirements, the questions we need answered, and how proposals will be evaluated.

Organization background

- [Describe your organization: industry, size, locations]
- [Regulatory or contractual requirements that apply]
- [Applications and teams that call LLMs today, and the providers, models and self-hosted models they use]
- [Monthly request and token volumes, and current LLM spend, if known]
- [Cloud platforms, network design and any private connections to cloud model services]
- [Identity provider, secrets manager, observability, SIEM and FinOps tools in use]
- [Data classification, data residency and AI governance policies that apply to model use]

2. Project Objectives

- Give every application one governed API for hosted and self-hosted models, so models can be changed without code changes
- Keep applications running when a model provider has an outage or limits our traffic
- Control and attribute LLM spend by team and application, with budgets that are enforced
- Apply consistent data protection, logging and guardrail policies to every model call
- Keep regulated data within approved regions and network boundaries
- Replace provider API keys held by individual applications with scoped, revocable gateway keys

3. Scope of Work

The selected vendor will be responsible for:

- Delivery of the gateway in the [vendor-hosted / customer cloud / self-hosted] deployment model, serving [regions]
- Connection to [list model providers and self-hosted model endpoints]
- Migration of [number] applications from direct provider integrations, in phases agreed with [Company Name]
- Configuration of routing, fallback, keys, budgets, rate limits, logging and guardrail policies
- Integration with our identity provider, secrets manager, observability platform, SIEM and FinOps tools
- Training for platform administrators and onboarding material for application developers
- Ongoing support and service management for the contract term

4. Requirements

For each requirement, enter a response code and a short explanation. Y = available today in the proposed edition; P = partially met or needs configuration/services (explain); R = on the roadmap (give a date); N = not supported. Must-have requirements are pass/fail; Nice-to-have requirements add points.

4.1 Functional

ID	Requirement	Priority	Response	Vendor comments
F1	One API endpoint through which our applications reach hosted model providers and our self-hosted models, compatible with a widely used chat-completions-style API and with the API formats our applications already use, so an application can change models through configuration rather than code.	Must		
F2	Support for chat and embedding requests, streamed responses, tool (function) calling and structured output. State which provider-specific features are passed through unchanged, which are translated into a common format, and which are not supported.	Must		
F3	Support for image, audio and document inputs, and for providers' batch processing interfaces, through the same API.	Nice		
F4	Routing rules based on application, team, model alias, request attributes or data classification, which administrators can	Must		

ID	Requirement	Priority	Response	Vendor comments
	change without redeploying applications.			
F5	Automatic retries and fallback to alternate models, providers, regions or self-hosted deployments on errors, timeouts or provider rate limits, with a fallback order we configure.	Must		
F6	Load balancing across several deployments, accounts or regions of the same model, and weighted traffic splitting so a new model or model version can be tested on a share of traffic before full rollout.	Must		
F7	Gateway-issued API keys for each team, application or project, scoped to approved models and budgets, with expiry and revocation. Provider credentials are held centrally and never exposed to applications.	Must		
F8	Token and spend budgets per team, application, project or key, with alerts at thresholds we set (by email, chat or webhook) and hard limits that block requests when a budget is used up.	Must		
F9	Rate limits on requests and tokens per minute, and concurrency limits, per key, team or application, returning error responses that applications can handle.	Must		
F10	Cost attribution for every request by team, application, project, user and model, using provider price lists (including differently priced token types such as cached input), our negotiated rates and our own cost rates for self-hosted models, with showback and chargeback reports we can export.	Must		
F11	Response caching for identical requests, with a time-to-live set per application, cache scoping per application or tenant, and a way for an application to bypass the cache on a request.	Must		
F12	Semantic caching that can return a stored response for a similar prompt, with a configurable similarity threshold, scoping by tenant and data permission, and cache invalidation.	Nice		
F13	A log record for every request: application, user, model, provider, tokens, latency, cost, status and policy decisions. Capture of full prompts and responses must be configurable per application, including an	Must		

ID	Requirement	Priority	Response	Vendor comments
	option to store no content at all.			
F14	Redaction or masking of sensitive data in prompts and responses before they are written to logs, and retention periods we configure per application.	Must		
F15	Input and output guardrails that detect personal data (PII) and custom patterns we define, such as account numbers or credentials, and redact, mask or block them, with policies set per application.	Must		
F16	Content filters for harmful content categories, with thresholds and actions (block, warn or log) set per application, provided natively or through integration with guardrail services we choose.	Must		
F17	Detection of prompt-injection and jailbreak attempts, in user input and in content returned by retrieval or tools, with block, warn or log actions.	Nice		
F18	Hooks or plug-ins that call our own or third-party guardrail and policy services before and after each model call.	Must		
F19	A model catalog that administrators govern: which models and versions are approved, for which teams, regions and data classifications, with model versions pinned per application. State how you tell us when a provider schedules a model we use for retirement.	Must		
F20	Dashboards for request volume, latency (including time to first token), error rates, token usage and cost by application, team, model and provider.	Must		
F21	Prompt templates stored in the platform, with versioning, approval before production use and rollback, which applications call by reference.	Nice		
F22	Evaluation hooks: sampling of production requests into evaluation datasets, scoring of live traffic (including model-graded checks), an API to record user feedback, and export to our evaluation tools.	Nice		
F23	Governance of agent tool use through the gateway, such as proxying Model Context Protocol (MCP) servers with authentication, tool allowlists and logging.	Nice		

4.2 Technical and architecture

ID	Requirement	Priority	Response	Vendor comments
T1	State which deployment models you offer: vendor-hosted SaaS, a data plane in our cloud account with a vendor-managed control plane, or fully self-hosted (state whether self-hosted deployment works in disconnected environments, with offline updates). For each, identify which components process or store prompt and response content, and what data or telemetry is sent to you.	Must		
T2	A gateway tier that scales horizontally with no single point of failure, deployable across availability zones and regions. State what happens to traffic if the control plane can't be reached.	Must		
T3	Added latency of no more than [X] milliseconds at the 95th percentile at [requests per second], with streamed tokens passed to the application as they arrive. State your measured figures with logging and guardrails on and off, and how you measured them.	Must		
T4	Data residency controls: requests from a given application or data classification are routed only to model endpoints in regions we approve, and logs are stored in a region we choose.	Must		
T5	Private connectivity between our applications, the gateway and cloud model services (private endpoints, peering or an equivalent), so model traffic does not have to cross the public internet. State the options for each cloud and deployment model.	Must		
T6	Support for self-hosted open-weight models behind inference servers with compatible APIs, with health checks and the same routing, limits, logging and guardrails as hosted models.	Must		
T7	Management of routes, keys, budgets and guardrail policies as code, through an API and an infrastructure-as-code or GitOps workflow, with promotion between development, test and production environments.	Nice		

4.3 Integration

ID	Requirement	Priority	Response	Vendor comments
I1	Single sign-on to the administration console with our identity provider via	Must		