Xtensyon.
CASE STUDY

Deploying LLMs On-Prem and in Private Cloud: Cost, Compliance, and Reliability Considerations

February 10, 2026 • Xtensyon Labs, Platform Engineering • 9 min read

> *Some teams cannot send sensitive data to third-party APIs, and network routes are not always predictable. This case study shares a private deployment pattern aimed at strong controls, clear cost reporting, and operational reliability.*

TL;DR

- Use a gateway for auth, quotas, and policy checks; do not rely on the UI.
- Plan for congestion with queues and fallbacks instead of timeouts.
- Track cost with per-team reports so GPU spend is explainable.
- Make model swaps a config change; you will rotate models over time.

Executive Summary

Not every enterprise environment is friendly to hosted LLM APIs. For teams dealing with data residency requirements, strict security reviews, or unreliable network routes, on-prem or private cloud deployment is often the practical choice. This case study summarizes an enterprise LLM service designed for predictable cost and reliability: controlled model choices, queue-based traffic shaping, tight observability, and clear fallbacks when capacity is constrained.

Why It Matters

Infrastructure choices shape adoption. If responses are slow or the service is frequently unavailable, users stop using it. Private deployments also introduce procurement and capacity planning realities.

A clean pattern makes the conversation easier: what you pay for, what you get, and what risks are controlled.

What We Built

- A multi-tenant LLM gateway with per-team quotas, request logging, and policy controls (PII redaction and prompt inspection).
- A two-tier serving strategy: a smaller default model for general use and a larger model for approved high-value workflows.
- Queue-based load management with graceful degradation (summarize to extract to route to human) when GPUs are saturated.
- End-to-end observability: latency, token usage, GPU utilization, cache hit rate, and error budgets tied to SLAs.

Observed Outcomes

- More stable latency under peak traffic by using queues and admission control instead of letting requests pile up.
- Lower spend surprises after implementing per-team quotas and monthly cost reports in business-friendly language.
- Easier security sign-off because data stays within the environment and model access is mediated by a gateway.

Implementation Notes

- Treat GPUs like shared infrastructure. Without quotas and priority rules, the noisiest team will starve everyone else.
- Invest in caching for repeated prompts and retrieval results. It is one of the simplest cost levers that does not reduce quality.
- Build for model swaps. You will change models; make it a config change, not a rewrite.
- Plan for incident response: define what gets turned off first (non-critical features) when capacity is tight.

Governance & Risk

- Security: keep prompts and outputs in protected logs; treat them as potentially sensitive data.

- Compliance: document where data is stored and how deletion requests propagate across logs, caches, and embedding stores.
- Operational risk: define SLOs for latency and availability, then align GPU capacity planning to those targets.

Release Checklist

- Do we have a gateway enforcing quotas, auth, and policy checks?
- Do we have observability that ties performance to user experience and cost?
- Do we have capacity fallbacks when GPUs are saturated?
- Can we rotate models without changing downstream applications?
- Is our logging and retention policy documented and auditable?

Conclusion

Private LLM deployment is not about chasing the biggest model. It is about guardrails around performance, security, and cost so the service remains usable month after month. If you build a gateway, enforce quotas, and prepare graceful fallbacks, you will avoid the common failure mode of a strong demo that becomes a painful platform to operate.

Keywords

on-prem llm | private cloud | data residency | enterprise security | GPU cost optimization | high availability | AI infrastructure