llmgw

Your own AI platform on premise: unlimited tokens on your own GPUs, one gateway in front of local models and cloud providers

Lower token cost · Data stays in-house · No single-vendor lock-in · Ready for agents, RAG and fine-tuning

Why on-premise AI, and what llmgw adds

1 Token cost keeps growing

Agents, MCP tools and coding assistants burn tokens; per-token cloud bills grow with every new use.

llmgw: Frequently used models run on your GPUs: no per-call meter, predictable cost. Usage is still counted per team and project for chargeback.

2 Some data must not leave

Engineering, production and personal data are not suited to cloud models.

llmgw: A project's keys can be held in the cluster: no outside provider, not even as a fallback. Guardrails catch PII and secrets.

3 No single vendor

Models and providers change every quarter; applications should not have to.

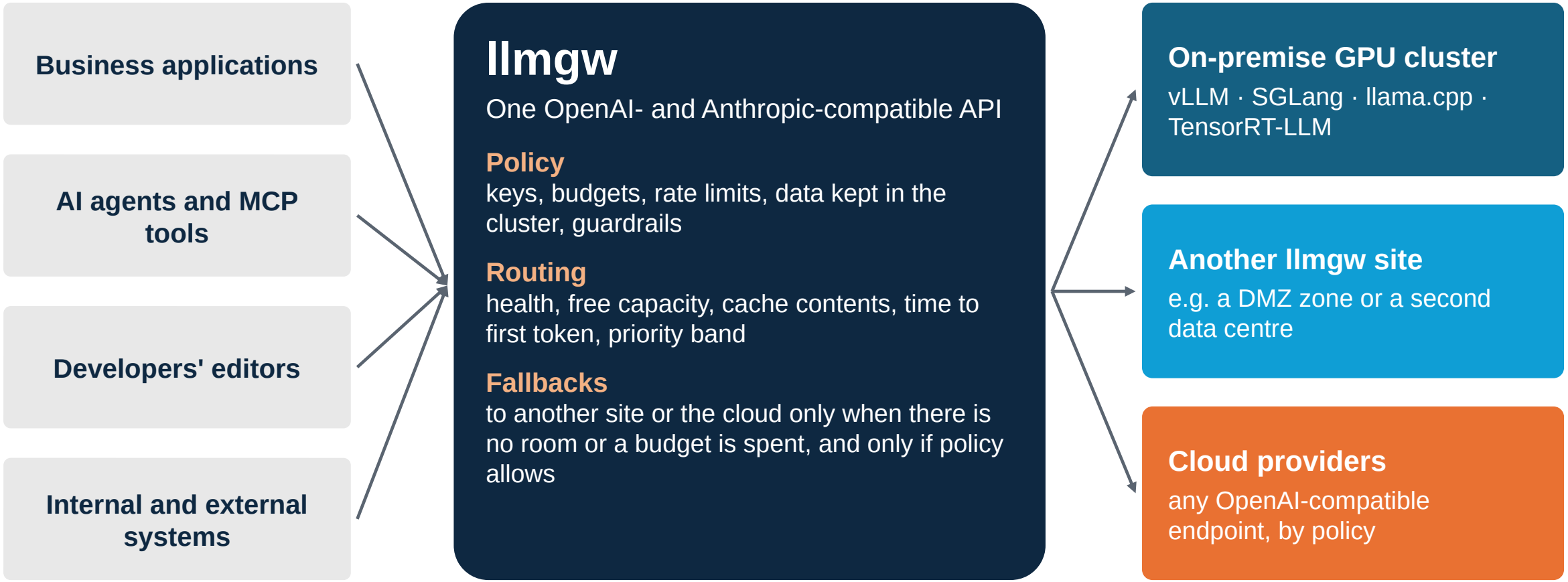
llmgw: One OpenAI- and Anthropic-compatible API over local open models, other llmgw sites and any OpenAI-compatible cloud endpoint.

4 A base for what comes next

RAG, fine-tuning, agents and multi-model work need one platform, not one per project.

llmgw: Knowledge bases, LoRA adapters, MCP tools, agent memory, picture and 3D jobs: all already in the gateway.

Apps ask for a model; llmgw decides where it runs



Swapping a model or a provider changes the catalog, not the applications.

Every request is placed by policy

Criterion	How llmgw meets it
Data classification and residency	Per-project 'in-cluster' keys never reach an outside provider; guardrails check for PII, secrets and prompt injection
Health	Agents on every node report GPUs and engines; failed replicas are replaced, nodes drained one replica at a time
Capacity	Admission control by priority band; routing by each replica's queue and free KV cache; models load on demand
Latency	Routing profiles: chat favours session affinity and fastest first token, batch the shortest queue
Cost	Budgets and rate limits per key, team and project; prices per token, image or mesh; usage export for chargeback
Resilience	Three control planes with a floating address; fallbacks to another site or provider

Unlimited tokens without giving up speed

0.5 ms

at most, added by the gateway at p99,

with authentication, usage accounting and pattern guardrails

1.24×

interactive p99 vs. idle while 500 batch requests run,

against 58× without admission control

125B

parameter mixture-of-experts model

served from a single GPU card (Strata engine)

On your own cards a token has no price per call: the cost is the hardware and power you already have. Interactive users keep their speed while agents and batch jobs fill the rest of the capacity.

Put the hardware you already own to work

1. **Any mix of GPUs**
   Setup picks engine, precision and workflow by each card's compute capability, from V100 and T4 to current data-centre cards.

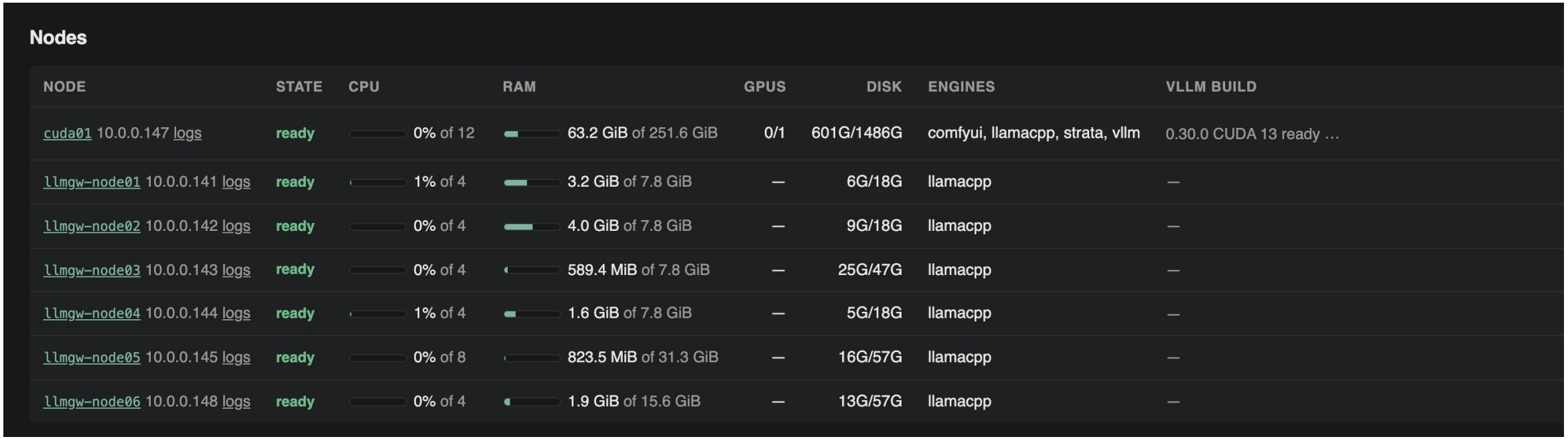
2. **Runs in virtual machines**
   Linux GPU VMs (RHEL-family 10, Ubuntu 22.04+, Debian 12+); our own reference cluster runs on GPU passthrough VMs.

3. **GPUs are shared, not reserved**
   Models load when someone asks and stop when nobody does; scaling windows per weekday keep a minimum where needed.

4. **CPUs count too**
   Guardrail judge, embeddings and small models run on CPU nodes, beside the GPUs.

5. **Models larger than one card**
   Replicas spread over several cards and machines; mixture-of-experts models fit a single card.

The cluster at a glance



Nodes

NODE	STATE	CPU	RAM	GPUS	DISK	ENGINES	VLLM BUILD
cuda01 10.0.0.147 logs	ready	0% of 12	63.2 GiB of 251.6 GiB	0/1	601G/1486G	comfyui, llamacpp, strata, vllm	0.30.0 CUDA 13 ready …
llmgw-node01 10.0.0.141 logs	ready	1% of 4	3.2 GiB of 7.8 GiB	—	6G/18G	llamacpp	—
llmgw-node02 10.0.0.142 logs	ready	0% of 4	4.0 GiB of 7.8 GiB	—	9G/18G	llamacpp	—
llmgw-node03 10.0.0.143 logs	ready	0% of 4	589.4 MiB of 7.8 GiB	—	25G/47G	llamacpp	—
llmgw-node04 10.0.0.144 logs	ready	1% of 4	1.6 GiB of 7.8 GiB	—	5G/18G	llamacpp	—
llmgw-node05 10.0.0.145 logs	ready	0% of 8	823.5 MiB of 31.3 GiB	—	16G/57G	llamacpp	—
llmgw-node06 10.0.0.148 logs	ready	0% of 4	1.9 GiB of 15.6 GiB	—	13G/57G	llamacpp	—

1. **CPU and GPU side by side**
   One GPU node and six CPU-only nodes in one cluster, each with the engines it can run.

2. **Live state of every machine**
   CPU, memory, GPUs and disk as the agents report them, seconds old.

3. **Engines installed for you**
   llama.cpp, vLLM, ComfyUI and Strata are installed by the node agent where they fit.

Three steps from empty servers to developers

1

GPU servers ready

- sudo llmgw sets up one machine, or three for high availability
- Nodes join over SSH: llmgw node add
- llmgw doctor checks GPUs and drivers before work lands
- Ends with a model answering

2

Hybrid consumption through one gateway

- OpenAI and Anthropic APIs on one endpoint
- Keys, budgets and limits per organisation, team and project
- Local first; cloud only as an allowed fallback
- Usage exported per project and model

3

The developer experience

- VS Code extension served by the gateway itself
- Continue, Codex and Cline connected from the account page
- Company MCP tools offered through the gateway
- Coding agents without token rationing

Agents and MCP tools without counting tokens

Tool-calling agents read, plan and retry.

A single task can take hundreds of model calls. With local models those calls cost capacity, not money, and priority bands keep people at the keyboard ahead of background agents.

Editors

VS Code extension from the gateway; Continue, Codex and Cline agent modes

Company tools via MCP

Internal MCP servers offered through the gateway, behind its keys and guardrails

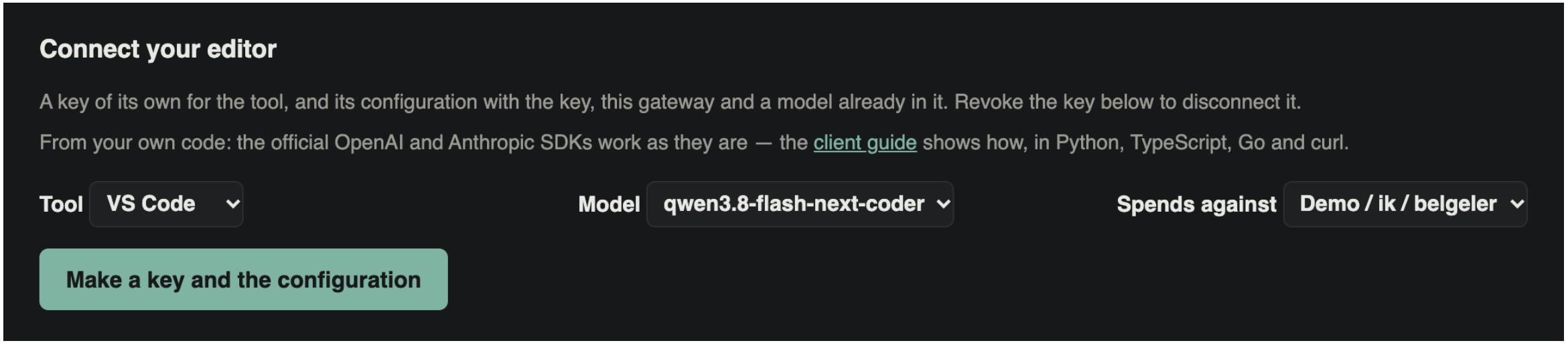
Memory and conversations

Agent memory and conversation state kept by the gateway

Long-running work

Batches, kept and replayed requests, ComfyUI workflows as jobs

Developers connect their editor in one step



1 **Choose the tool**
VS Code, Continue, Codex or Cline; the gateway writes the configuration.

2 **Choose the model**
Any model the person may use, local or by policy remote.

3 **Choose who pays**
The key spends against a project's budget, so usage lands where it belongs.

Fits your API management and network zones

Behind your API management

Sits behind an enterprise API gateway (Kong documented). Teams keep their own llmgw keys, so budgets, guardrails and usage stay per team.

Separate zones, separately governed

An internal cluster and a DMZ cluster each run their own policy; one can use the other as a provider over a private link.

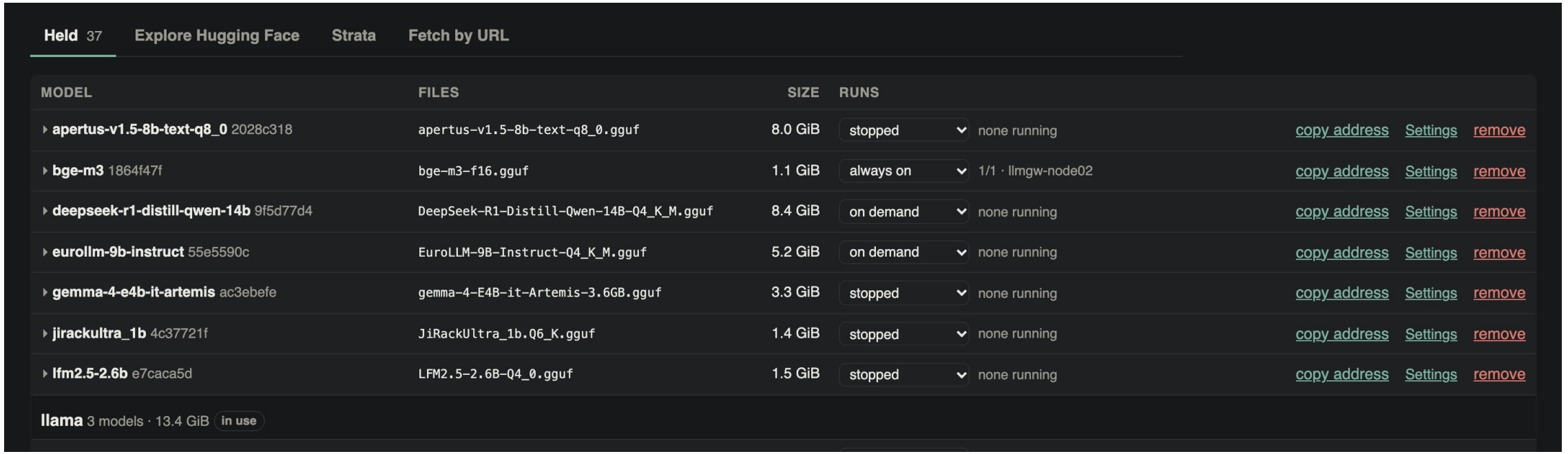
Hardened edge

TLS on every internal port, Caddy or nginx in front of the client port, a firewall for llmgw's ports, admin port kept apart.

Air-gapped if needed

Offline installs from a bundle; model weights from an internal mirror, pinned to a commit and checked.

Model weights from an internal mirror



1. **Pinned and verified**
   Each model is held at a commit and checked against what Hugging Face published.

2. **Run mode per model**
   Stopped, on demand or always on, changed from this page.

3. **One way in**
   New models come in through Explore Hugging Face or Fetch by URL; nodes load weights only from the mirror.

Security and governance built in, not bolted on

Identity

Single sign-on (OpenID Connect), Active Directory / LDAP, SCIM provisioning

Tenancy

Organisations, teams, projects; row-level security in PostgreSQL

Guardrails

PII, secrets, prompt injection, deny patterns, a judge model; observe or enforce

Audit

Every change logged; viewer, operator, billing and admin roles

Privacy

Projects held in the cluster; a person can be forgotten on request

Supply chain

Signed releases; weights pinned and verified; secrets in Vault

Guardrails record decisions, never content

Recent

WHEN (UTC)	REQUEST	WHERE	MODEL	ACTION	POLICY	CATEGORY	STATUS	MS
2026-10-31 22:01:06	req_fb7c3f2797a016347981e01b	personal	agent	shadow-flagged	observe	injection · injection_template	503	8
2026-10-31 21:46:23	req_ce85c83365395066e5cd30ba	personal	agent	shadow-flagged	observe	injection · injection_template	503	3
2026-10-31 21:45:36	req_f53e8f2306d45ba6b3d09305	personal	agent	shadow-flagged	observe	injection · injection_template	503	3
2026-10-31 21:18:54	req_542046fa4e087a27e764f890	personal	qwen3.8-flash-next-iq2_xs	shadow-flagged	observe	injection · injection_encoded	200	82
2026-10-31 21:16:56	req_757f47ac03412e37c4458714	personal	qwen3.8-flash-next-iq2_xs	shadow-flagged	observe	injection · injection_encoded	200	70
2026-10-30 18:47:24	req_483f8ea8ead21976dd7c446f	Demo / ik / belgeler	bench-sim	shadow-redacted	observe	email · pii:email	200	1
2026-10-30 18:47:24	req_909e21f03de85664f40fde77	Demo / ik / belgeler	bench-sim	shadow-redacted	observe	email · pii:email	200	1
2026-10-30 18:47:24	req_86c17fe0b785bdd1058d4c91	Demo / ik / belgeler	bench-sim	shadow-redacted	observe	email · pii:email	200	1

1

Redacted on the way

E-mail addresses in a project's prompts were masked before reaching the model.

2

Injection flagged

Prompt-injection attempts against an agent were caught and recorded.

3

Observe first, then enforce

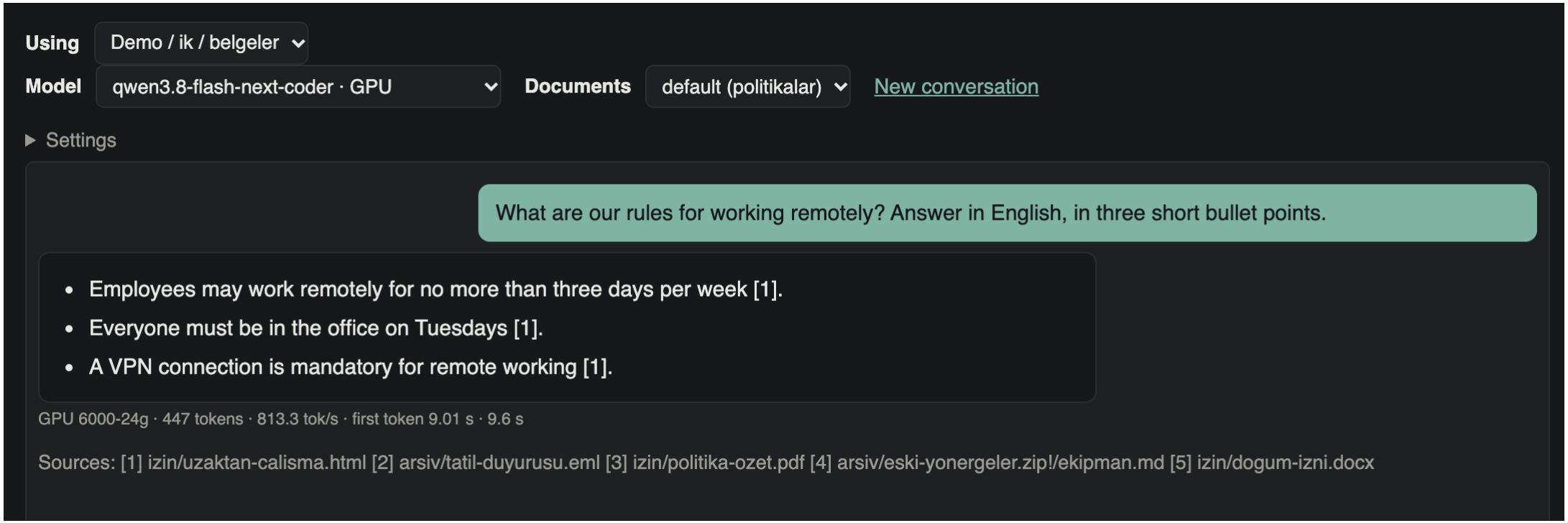
Each policy can watch before it blocks; categories and counts only, no content.

A foundation that grows with the AI roadmap

RAG	Fine-tuning	Agents	Multi-model	Beyond text
Knowledge bases fed from people's own machines (PDF, Office, mail); answers with sources; kept apart per tenant	LoRA adapters trained from a collection on spare GPU memory, loaded on demand	MCP tools, agent memory, Anthropic Messages API with reasoning	Canaries with automatic rollback, quality suites gating updates, response cache	Pictures, 3D models and any ComfyUI workflow as jobs on the same cards

One platform: the same cards, keys, budgets and guardrails for every new use case.

Answers from your own documents, with sources



1

Your documents, any language

The policies are Turkish HTML, PDF, DOCX and e-mail; the question and answer are English.

2

Every claim has a source

Answers cite the documents they came from, numbered.

3

On your own GPU

447 tokens at 813 tok/s on one 24 GB card; nothing left the cluster.

Production grade from the first install

0

requests lost when the database primary lost power

Three machines each run the control plane with a floating address; any one can be lost.

Run from one screen

Bare llmgw opens a console for every task; the same actions on the command line and the admin API

Observe

Prometheus metrics, Grafana dashboards, Alertmanager and signed webhooks, tracing to the engines

Change safely

Settings in force within seconds on every control plane; cluster upgrades that report progress; encrypted backups

Automate

Admin API in OpenAPI, a Terraform / OpenTofu provider, a Helm chart for the control plane

How llmgw compares

	llmgw	LiteLLM	Kong AI Gateway	Agent Router (Envoy AI Gateway)	llm-d	NVIDIA Dynamo
Runs models on your own GPUs	**Yes: places, starts, stops**	No: proxy only	No: proxy only	No: routes only	Yes	Yes
Cloud providers behind one API	OpenAI-compatible ones	Yes, 100+	Yes	Yes	No	No
Keys and budgets per team	**Yes**	Yes (some paid)	Token limits (paid)	Quotas, usage	—	Left to a gateway
Cache-aware routing	**Yes (vLLM prefix cache)**	—	—	Via InferencePool on K8s	Yes	Yes
Guardrails: PII, injection	**Yes, built in**	Yes (several paid)	Yes (paid)	—	—	—
Knowledge bases	**Yes: ingest client, per tenant**	Via vector stores	RAG Injector (paid)	—	—	—
Runs without Kubernetes	**Yes: one binary**	Yes: Docker	Yes: Docker	Standalone or K8s	No: K8s	K8s for production

Gateways route to models that someone else runs. Serving stacks run models but leave keys, budgets and guardrails to you. llmgw does both, without Kubernetes.

From each project's public documentation, November 2026. Paid = enterprise licence. — = not in their documentation.

Gateways route; llmgw also runs the models

LiteLLM
Open-source proxy and Python SDK

Strong at

- 100+ providers behind an OpenAI-compatible API
- Virtual keys, budgets and spend per key, team and user
- Admin UI, fallbacks, load balancing

Leaves to you

- Running the models: it proxies existing endpoints
- SSO beyond 5 users, audit logs, per-team guardrails: Enterprise licence

llmgw adds

Places and runs the models on your GPUs; SSO, SCIM and the audit log included

Kong AI Gateway
AI plugins on Kong Gateway

Strong at

- API management many companies already run
- Many providers, semantic cache, prompt guard, MCP proxy

Leaves to you

- Running the models
- Token rate limiting, PII sanitizer, RAG injector: AI Gateway Enterprise
- Open source ends at Kong 3.9

llmgw adds

Keys, budgets, guardrails and knowledge bases in the same binary

Agent Router
formerly Envoy AI Gateway

Strong at

- Envoy-based routing and failover
- Quotas and usage attribution per identity
- MCP access control; InferencePool on Kubernetes

Leaves to you

- Running the models
- Its full feature set through Kubernetes resources

llmgw adds

One binary on VMs or bare metal; HA from three machines

Each of these can still sit in front of llmgw, and a site that already runs one keeps it.

From each project's public documentation, November 2026.

Serving stacks run; llmgw also governs

llm-d
CNCF Sandbox, Kubernetes-native

Strong at

- Prefix-cache-aware routing, KV cache offload
- Prefill/decode on separate workers over RDMA
- NVIDIA, AMD, Intel Gaudi, Google TPU

Leaves to you

- A Kubernetes cluster to run it on
- Keys, budgets, guardrails, a UI: not in its docs
- Cloud providers

llmgw adds

Runs on plain VMs; keys, budgets and guardrails per team

NVIDIA Dynamo
Datacenter-scale inference, Apache 2.0

Strong at

- KV-aware routing; disaggregated prefill and decode
- vLLM, SGLang and TensorRT-LLM backends

Leaves to you

- Kubernetes for production
- Auth and rate limiting: a gateway at the edge
- Focus on NVIDIA hardware

llmgw adds

The gateway built in, with a portal and editor setup for developers

vLLM production-stack
Reference stack for vLLM

Strong at

- Kubernetes deployment of vLLM at cluster scale
- KV-, load- and prefix-aware routing
- Shared KV cache, metrics dashboard

Leaves to you

- vLLM only
- Keys, budgets, guardrails: not in its docs

llmgw adds

vLLM, SGLang, llama.cpp and TensorRT-LLM under one API

llmgw's precise KV routing and prefill/decode disaggregation are on its v1.0 roadmap.

From each project's public documentation, November 2026.

Works alongside the gateway you already run

Gateway in front of llmgw

Tested with Kong Gateway 3.9

- The site's API policy stays: llmgw is one more upstream
- A plain route, or Kong's ai-proxy plugin for its own token statistics
- Each team keeps its own llmgw key, so budgets, limits, guardrails and usage stay per team
- Bills from llmgw's usage export, which counts streamed answers too

Gateway behind llmgw

For cloud providers the site already reaches

- Any OpenAI-compatible gateway becomes an external model in llmgw
- As a fallback, used only when the cluster has no room
- The provider keys stay where they are; llmgw holds one consumer key
- Requests from projects marked in-cluster never take this path

llmgw needs no outside gateway for keys, budgets, guardrails or routing to its own replicas.

Kong in front tested in DB-less mode; Kong behind documented, not yet tested.