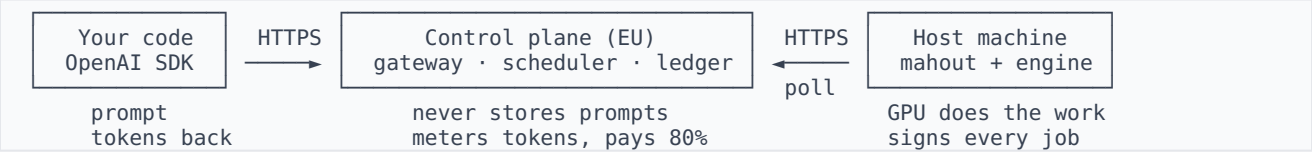
How ElephantPool works

A distributed inference network: open-weight models served by community GPUs across Europe, behind one OpenAI-compatible API. The whole path, from an API call to a signed receipt.

1. The shape of the system

Three parties, one contract between them.



The host **polls outward**; nothing connects in. That single decision is why a GPU behind a home router works with no configuration, and why a host never exposes a port to the internet.

2. What happens to one request

1. **Your call arrives** at the gateway with an API key. We check the key, estimate the cost, and confirm your prepaid balance and the key's spend cap cover it.
2. **The job is queued** for the model you asked for. It carries your messages and nothing about you.
3. **A host claims it** — one that has proven it can serve that model, by benchmark, on that machine.
4. **The GPU produces tokens**, streamed back through the gateway to you as they are generated.
5. **The job settles.** Tokens are counted, your balance is debited to the micro-cent, the host is credited 80%, and the prompt is dropped from the record. What remains is counts and timestamps — no text.
6. **The host signs a receipt** with a key that never leaves its machine: model, exact build, token counts, timing.

Prompts live in memory for the duration of the job. They are not written to disk, not logged, and not used for training — by us or by hosts.

3. mahout — the agent on the host

mahout is one open-source Rust binary, roughly 5 MB, named after the person who rides the elephant. It runs as an ordinary user, needs no root, and does six things:

- **Probes** the machine — CPU, RAM, GPUs and their VRAM.
- **Enrolls** with a one-time code from the dashboard, generating an ed25519 identity locally; the private key never leaves the machine.
- **Fetches and verifies** model weights and the engine into a content-addressed cache — checked against pinned SHA-256 digests while streaming and again at every load.
- **Supervises the engine**: starts it, watches its health, restarts it once if it dies, stops it cleanly.

- **Polls and serves** jobs, streaming tokens back in batches.
- **Signs a receipt** per job, appended to a local log you can re-verify offline with `mahout receipts verify`.

It pauses itself while you are using the machine, and `mahout uninstall` prints a deletion plan before removing anything. The source and the installer are public.

4. Models are builds, not files

A *model* is what a customer requests (`qwen3.8-27b`). A *build* is a concrete artifact: one quantization, one engine, one exact file from one exact upstream revision, with the SHA-256 published for it.

That split is what lets a single model reach every class of GPU. Qwen3.8 27B ships in five builds, from Q5_K_M for a 24 GB card down to IQ2_XXS for an 8 GB one. The customer's request never changes; the host picks the build its card can actually run.

Card	Build it serves	On disk
24 GB	Q5_K_M — reference quality	18.4 GB
16 GB	Q3_K_XL — balanced	12.2 GB
12 GB	Q2_K_XL — compact	9.2 GB
8 GB	IQ2_XXS — compact	6.8 GB

A build only becomes claimable on a machine after that machine has **loaded and benchmarked it**. Estimates decide what is worth downloading; measurements decide what is worth routing.

5. Engines — and the container question

mahout does not do the arithmetic itself. It drives an inference engine, and the engine belongs to the *build*, not to the host — which is what makes adding an engine an additive change rather than a rewrite.

Engine	Shape	Role	Status
llama.cpp	~5 MB binary, no container, no root	The consumer tier. One Vulkan build serves AMD, NVIDIA and Intel alike.	In production
vLLM	10–30 GB container image, GPU toolkit	The workstation and datacenter tier: continuous batching at scale, paged attention, native AWQ/FP8, multi-LoRA.	Planned

So what are the images, exactly?

Today, on a host machine: **none**. There is no Docker image to pull, no daemon to install, no root to grant. A 5 MB binary and a checksummed model file is the entire footprint. We think that is the right shape for a gaming PC in a living room, and we did not want to pretend otherwise for the sake of sounding cloud-native.

Containers matter at the other end of the market. When the vLLM tier ships, engine images will be **upstream images referenced by digest** — `vllm/vllm-openai@sha256:…`, not a moving tag — and

the list of images a host may run is **compiled into the mahout release itself**. The control plane can tell your machine *which* allowed image to start; it can never introduce a new one. The container gets no network egress and a read-only mount of the verified weights.

Three rules govern that tier, and they are the same rules that govern model weights:

- **Pinned by digest.** A digest is immutable and verified by the container runtime; a tag is a promise someone else can change.
- **Allow-listed in the client.** A host runs what its own binary permits, not what a server asks for.
- **Opt-in per host.** Nobody wakes up hosting a new engine.

And the registry?

For upstream engines we do not host images at all — re-hosting a public image adds terabytes of storage and no security, because the digest already proves what you got. We run our own registry for exactly one case: images **we** build. The first of those is the verification hook — a build of the engine that emits a cryptographic commitment to its own activations, so a job's output can be checked rather than trusted. That image has to be ours, so it will be published from our own EU registry, signed and pinned by digest like everything else.

In short: no images on hosts today; upstream images pinned by digest when the vLLM tier lands; our own registry only for images we produce ourselves.

6. What is verified, and by whom

Artifact	Pinned as	Verified by
mahout binary	SHA-256 published next to the download	the installer, before it installs
Inference engine	SHA-256 compiled into mahout	mahout, on first use
Model weights	upstream revision + SHA-256 from the LFS pointer	mahout, while streaming and at every load
Engine images (future)	image digest, allow-listed in the release	the container runtime
Completed jobs	ed25519 signature by the host key	anyone, offline, from the receipt

A failed check is not a warning: the artifact is discarded and the job does not run. We would rather a host serve nothing than serve something we cannot name exactly.

7. Money

Customers pay per token, in euros, per model, with no subscription and no minimum: you top up a prepaid balance, and every API key carries a spend cap so a runaway loop cannot drain it. Hosts receive **80%** of the revenue from the jobs they serve, paid by SEPA transfer from €50.

The dashboard quotes a host's earnings as euros per *serving* hour, computed from the decode rate their own card measured — not from a projection, and never as a monthly figure, because that would require promising demand we cannot promise.

8. Where we are honest about limits

- The engine currently runs as an unprivileged child process bound to localhost, not inside a syscall sandbox. Hardening (seccomp, Landlock, no-egress containers) is planned and not yet done.
- Linux x86_64 is the only released host platform. Windows and macOS are portable in principle and not shipped in practice.
- Output verification (activation commitments) is designed and not yet enforced; today's guarantees are receipts and reputation.

The technical specification distinguishes throughout between what runs today and what is designed. So does mahout's own `AGENTS.md`, in the public repository.

Start

Build: point your SDK's base URL at `https://app.elephantpool.ai/api/gateway/v1` and pass a model id from the catalogue.
Host: `curl -fsSL https://elephantpool.ai/install-mahout.sh | sh`, then link the machine from your dashboard.

Model catalogue · Knowledge base · Download mahout · Source code · Whitepaper · Technical specification