TECHNICAL SPECIFICATION · V0.1

mahout & the Elephant Pool control plane

Engineering specification of the distributed inference network: host agent, control plane services, wire protocol, scheduling, verification, and the Phase 0 build cut.

NVIDIA + AMD day one | TOPLOC-verified | OpenAI-compatible | EU control plane

Contents

This is the engineering-level specification of the two systems that make the network run: the **control plane** (our servers) and `mahout` (the host executable). Wire formats, state machines, schemas, and algorithms are specified concretely enough to start building Phase 0. Design constraints inherited from WHITEPAPER.md: one model instance per host (no WAN sharding), request-level distribution, measured-not-declared capabilities, TOPLOC receipts, zero-retention data path, fiat rails + BSV anchoring.

---

0. Terminology & invariants

Term	Meaning
host	A machine running `mahout`, identified by an Ed25519 keypair
node	A schedulable engine instance on a host (a host with 2 independent GPUs may run 2 nodes)
model build	A concrete servable artifact: base weights + quantization + engine image, content-addressed
job	One inference request assigned to one node under a lease
receipt	Signed statement by a host that a job was executed: token counts + TOPLOC commitment
epoch	Payout accounting period (1 hour)

Global invariants:

1. **I1 — No inbound connectivity required on hosts.** Everything `mahout` does is outbound (NATS, QUIC, HTTPS). Works behind CGNAT.
2. **I2 — Hosts never receive executable code.** Only data: signed engine container images (from a fixed allowlist), signed weights, prompts.
3. **I3 — Prompts/completions exist only in RAM** on gateway and host. Disk spill is a build-breaking bug. Logs carry IDs and counts, never content.
4. **I4 — Every billable token is covered by a signed receipt**, and every receipt is covered by a Merkle anchor within 10 minutes.
5. **I5 — The control plane can crash without corrupting money**: billing state is derived from the receipt log (append-only), never from in-memory counters.

---

1. System topology

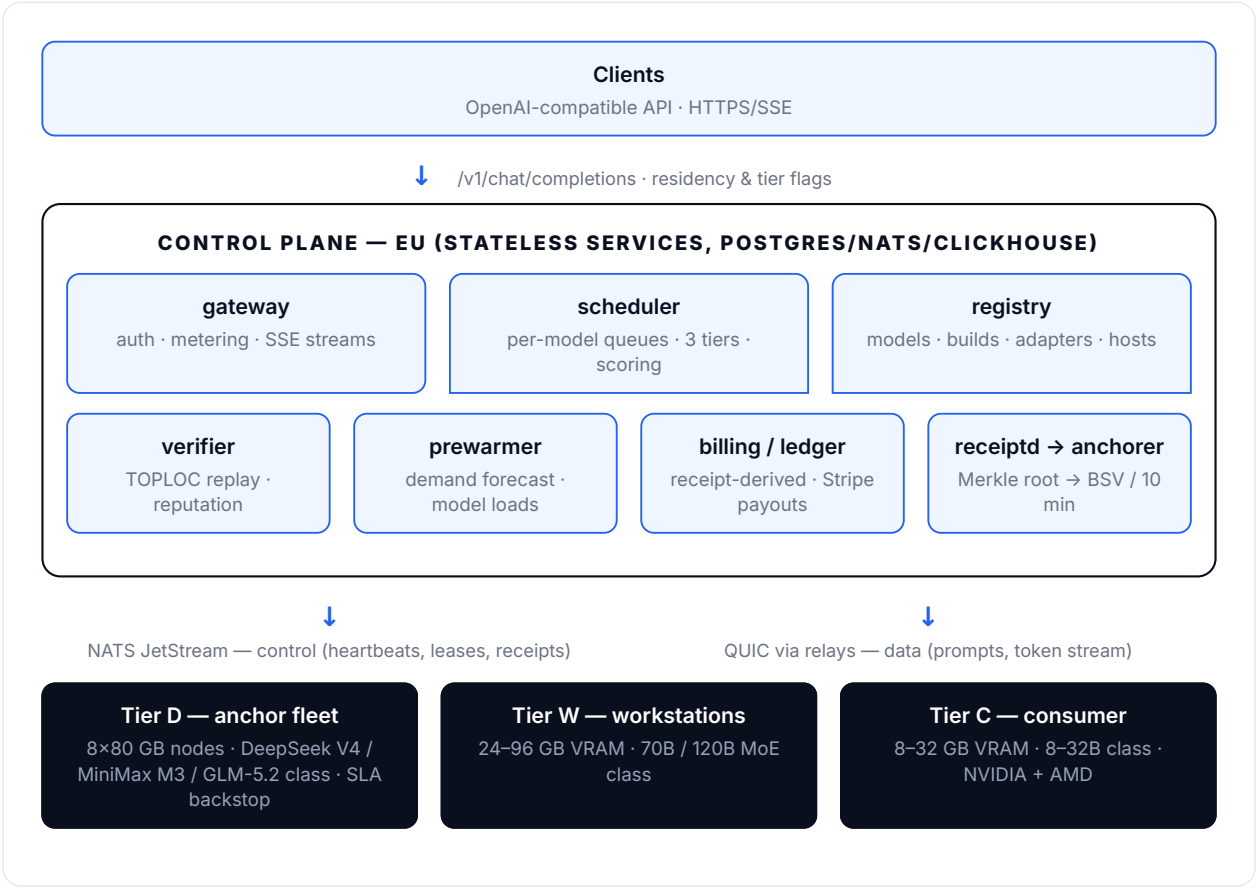

Figure 1 — Network topology: request-level distribution; a model instance never spans hosts over WAN.

Two control-plane regions (EU): `eu-west` (primary, BE/NL), `eu-central` (DR). Postgres with synchronous replica in-region + async cross-region; NATS JetStream cluster of 5 (3+2); gateways and relays are stateless and horizontally scaled behind anycast/GeoDNS.

2. Identity, enrollment, and transport security

2.1 Host identity

- On first run, `mahout` generates **Ed25519 keypair** `(host_sk, host_pk)`; `host_id = base32(sha256(host_pk)[0..20])`.
- Enrollment: `mahout up` → opens browser to `https://pool.elephantpool.ai/link?code=XXXX` → user authenticates → registrar binds `host_pk` to the account and issues:
  - a **host certificate** (X.509, our internal CA, 30-day lifetime, auto-renewed at half-life) embedding `host_id` and account id — used as TLS client cert for NATS and QUIC;
  - initial **NATS credentials** (JWT, NKey) scoped to that host's subjects only (see §4.1).
- Key rotation: new key signed by old key, submitted to registrar; old key valid 24h for overlap.

- Compromise recovery: account owner revokes host from dashboard → cert to CRL, NATS JWT revoked, pending payouts frozen pending review.

2.2 Transport

Channel	Protocol	Auth
Control (heartbeat, leases, config)	NATS 2.x over TLS 1.3	NATS JWT + TLS client cert
Data (prompt/tokens)	QUIC (quinn), ALPN `ep-data/1`	TLS 1.3 mutual, client cert = host cert; gateway cert pinned in `mahout`
Bulk (weights, images)	HTTPS to CDN + P2P (§7.3)	signed URLs; content addressing makes tampering moot
Enrollment/API	HTTPS	OAuth session / API keys

QUIC connection establishment for a job: the **gateway dials the relay**, the **host maintains a standing QUIC connection to the same relay** (outbound, satisfies I1), and the relay stitches streams by `host_id`. Optional direct path upgrade: relay coordinates QUIC hole punch (both sides simultaneous open); on success traffic migrates off-relay (QUIC connection migration), relay stays as fallback. Relays never see plaintext: the gateway↔host stream carries an inner TLS session (double encryption on relayed segments; the ~10–15% overhead is acceptable, and the direct path drops it).

---

3. Control plane services

All services are Rust (axum/tonic), deployed as containers, one binary per service. Postgres 16 is the source of truth for slowly-changing state; JetStream for queues; ClickHouse for high-volume telemetry; Redis only as gateway-local cache (no correctness dependency).

3.1 `gateway`

- Terminates client HTTPS. OpenAI-compatible surface:
  - `POST /v1/chat/completions` (streaming SSE + non-streaming)
  - `POST /v1/completions`, `POST /v1/embeddings`
  - `GET /v1/models` (from registry, with per-model pricing metadata extension `x-elephant`)
  - Elephant extensions in the request body: `service_tier: "interactive"|"standard"|"batch"`, `data_residency: "any"|"eu-sovereign"`, `max_price` (€/Mtok cap → router constraint), `session_id` (affinity hint).
- Validates API key (local cache, 5s TTL on limits), estimates prompt tokens (registry-shipped tokenizer, exact count comes from engine later), checks balance ≥ estimated cost × 1.5, then submits a **job envelope** to the scheduler and awaits assignment (§5).
- Owns the client SSE connection; pipes tokens from the host QUIC stream; buffers the last flushed offset per job for mid-stream failover replay (§5.5).
- Stateless: any gateway can pick up a retried job; in-flight jobs die with their gateway (client sees a retryable 502 before first token; after first token, failover per §5.5).

3.2 `scheduler` (sharded per model)

One logical scheduler; internally sharded by `model_id` (consistent hashing over scheduler replicas) so hot models don't block cold ones. Holds in memory, per model:

- **queue** of waiting job envelopes, one priority sub-queue per service tier;
- **node table**: every node currently advertising this model — state (`warm` , `loading` , `cold-capable`), engine slots free, EWMA stats, price bid, reputation, region flags;
- **affinity map**: `session_key → node_id` with TTL (default 15 min sliding).

Rebuilt from NATS + registry on restart in <5s; nothing durable lives here (I5).

3.3 `registry`

CRUD + read-mostly serving of: model catalog, model builds, engine image digests, LoRA adapters, host/node inventory, price book, eval gates. Everything versioned; every artifact row carries `sha256` , `sig` (our release key), `size` , `uri[]` (CDN + torrent infohash). Postgres, with a signed JSON snapshot (`catalog.json` , re-signed on change) served via CDN — `mahout` consumes the snapshot, not the DB.

3.4 `metering` → `billing`

- Gateway emits `usage_event` per job completion (or per 30s for long streams) → ClickHouse (analytics) **and** the receipt log (billing truth, I5).
- `billing` folds receipts hourly into: client debits (price book at job submit time, recorded in the envelope), host credits (payout book), platform margin. Client balances in Postgres with serializable transactions; payout ledger append-only.
- Payouts: monthly SEPA batch (threshold €50) via PSP API; the ledger stores PSP transfer ids for reconciliation. (Entity/PSP structure per the regulatory doc — the control plane just needs the ledger to be exportable per legal entity.)

3.5 `verifier`

Consumes the receipt stream; schedules re-execution of a sample (§8); writes `verification_event` s; feeds `reputation` .

3.6 `prewarmer`

Per-model control loop (30s tick), see §6.2.

3.7 `receiptd` + `anchorer`

`receiptd` validates receipt signatures against host certs, appends to the receipt log (Postgres partitioned by day + object-storage archive). `anchorer` every 10 min: Merkle tree over new receipt hashes → root into an OP_RETURN tx via ARC → stores txid + BUMP proof; receipts get `anchor_id` . Public verification endpoint: `GET /v1/receipts/{id}/proof` returns receipt + Merkle path + BUMP (SPV-verifiable offline).

3.8 `relay` fleet

Dumb QUIC stream stitchers, ~5 PoPs EU (scalable to global). Stateless; host→relay assignment by latency probe at `mahout` startup, re-probed hourly. Sized ~1 vCPU per 2k concurrent streams (relayed segments are just encrypted byte shuffling).

---

4. Wire protocol

4.1 NATS subject space

Host-scoped (host JWT only permits these):

```
ep.host.<host_id>.hb              → heartbeat (core NATS, 10s interval)
ep.host.<host_id>.state           → node state transitions (JetStream)
ep.host.<host_id>.cmd             ← commands: load/unload model, rebench,
                                     update, drain, set-config   (JetStream, durable)
ep.host.<host_id>.lease           ← job leases offered to this host (core, TTL'd)
ep.host.<host_id>.receipt         → signed receipts (JetStream, ack-required)
```

Control-plane internal:

```
ep.sched.<model_id>.enqueue       gateway → scheduler
ep.sched.<model_id>.assign        scheduler → gateway (reply-to pattern)
ep.reg.events                     registry change feed (catalog invalidation)
ep.verify.tasks                   verifier work queue
```

4.2 Heartbeat (host → control, every 10s, protobuf)

```
message Heartbeat {
  string host_id = 1;
  uint64 seq = 2;
  repeated NodeStatus nodes = 3;      // per engine instance
  HostVitals vitals = 4;              // cpu%, ram, vram free/total per GPU,
                                      // disk free, net up/down estimate, temps
  uint32 policy_state = 5;            // ACTIVE | PAUSED_USER | PAUSED_GAME |
                                      // PAUSED_WINDOW | DRAINING
}
message NodeStatus {
  string node_id = 1;
  string model_build = 2;             // sha256 of loaded build, "" if empty
  repeated string loaded_adapters = 3;
  uint32 slots_total = 4;             // max concurrent sequences (from bench)
  uint32 slots_busy = 5;
  float  toks_per_sec_ewma = 6;
  float  ttft_ms_ewma = 7;
  EngineKind engine = 8;              // VLLM | LLAMACPP
}
```

Three missed heartbeats (30s) → node marked `suspect`, no new leases; 60s → `down`, in-flight jobs failed over (§5.5).

4.3 Job lease (control → host)

```
message JobLease {
  string job_id = 1;
  string model_build = 2;
  repeated string adapters = 3;        // LoRA ids to apply
  string gateway_hint = 4;             // relay + gateway route token
  bytes  stream_ticket = 5;            // one-time token the gateway must present
                                       // on the QUIC stream to claim this job
  uint64 deadline_unix_ms = 6;         // lease expiry (accept + TTFT by then)
  Tier   tier = 7;
  uint32 max_output_tokens = 8;
  uint32 prompt_tokens_est = 9;
}
```

Host replies `ACCEPT` / `REJECT(reason)` within 500 ms on the same subject (reply-to). Non-response = reject. Accepting reserves one slot.

4.4 Data stream (gateway ↔ host, one QUIC bidi stream per job)

Framed with 4-byte length prefix + 1-byte type, CBOR payloads:

```
G→H  JOB_OPEN     {job_id, stream_ticket, params: {messages|prompt, sampling,
                   stop, tools, response_format}, tier}
H→G  JOB_META     {prompt_tokens_exact, engine, model_build, started_at}
H→G  TOKENS       {seq, text_delta, logprobs?}          // batched ~20–50ms
H→G  JOB_DONE     {completion_tokens, finish_reason, toploc_commitments[],
                   timing: {ttft_ms, total_ms}}
H→G  JOB_ERR      {code, detail}                        // OOM, engine crash…
G→H  CANCEL       {reason}                              // client disconnect
```

The host also emits the **receipt** (§8.1) on NATS after `JOB_DONE` — the QUIC path is for latency, the NATS path is for money; the gateway independently reports its own view (token counts, timings) and `receiptd` cross-checks the two before the receipt becomes billable (mismatch >1% → verification task, not payment).

---

5. Scheduling

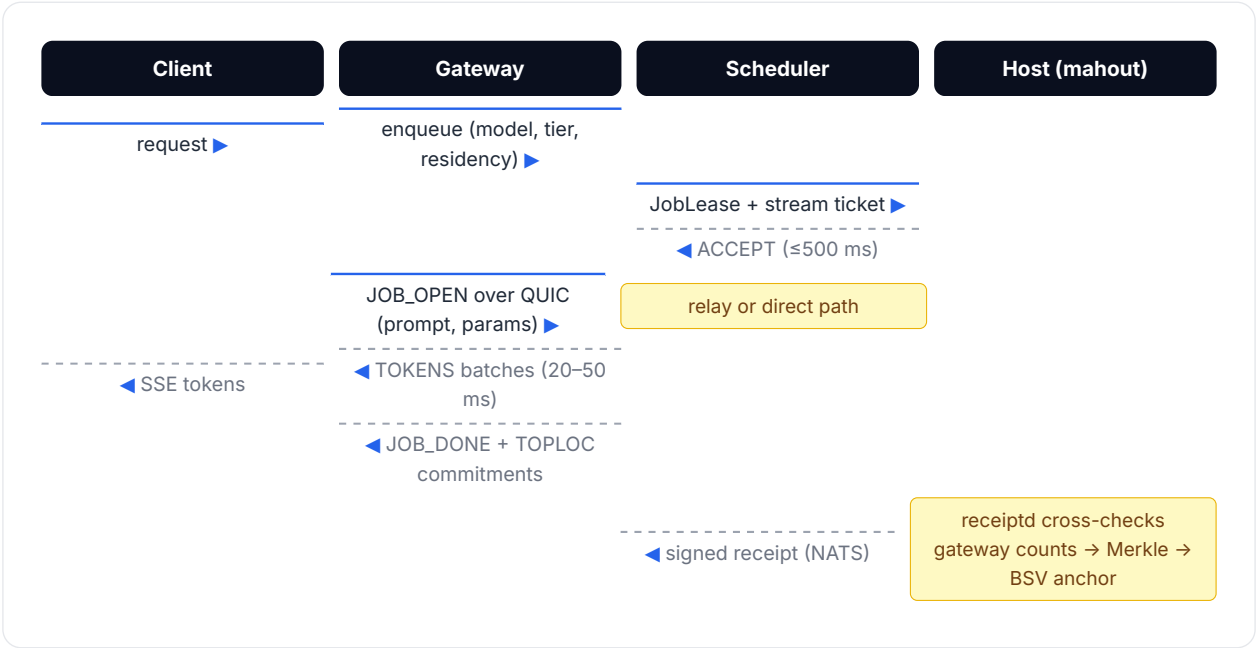

Figure 2 — Job lifecycle: latency path (QUIC) and money path (signed receipts) are separate channels.

5.1 Admission (gateway)

1. Resolve model alias → `model_id` (+ pinned build unless client pinned one).
2. Constraints from request: tier, residency, `max_price`, adapters.
3. `ep.sched.<model_id>.enqueue` with the envelope; await assignment with tier-dependent timeout (interactive 5s, standard 120s, batch: immediately ack `202` + job id, deliver via webhook/poll).

5.2 Dispatch loop (scheduler shard, per tick and on every state change)

```
for tier in [interactive, standard, batch]:
  for job in queue[tier] (FIFO within tier):
    C ← nodes where: model warm (or tier≠interactive and warm-capable),
                     slots free, residency ok, adapters loadable,
                     price ≤ job.max_price, reputation ≥ tier_floor
    if C empty: leave in queue; signal prewarmer
    score(n) = 0.30·norm(toks_per_sec_ewma)
             + 0.20·norm(1/ttft_ms_ewma)
             + 0.20·reputation                    // [0,1], §8.3
             + 0.15·affinity(job.session_key, n)  // 1 if pinned else 0
             + 0.10·(1 - price_bid/price_ceiling)
             - 0.05·(slots_busy/slots_total)
    offer lease to argmax; on reject/timeout try next-best (max 3), then requeue-front
```

Weights are config, tuned per model from telemetry. Anti-starvation: a job's effective tier escalates one level after 2× its SLA budget in queue.

5.3 Batch tier as filler

Batch jobs are dispatched only to nodes below 60% slot occupancy, and are the designated workload for: newly enrolled hosts (reputation ramp), model-load warmup validation, verification replays, LoRA training shards (§9), and synthetic evals. This is what guarantees "a new host earns something in week one" without exposing clients to unproven nodes.

5.4 Leases & timeouts

Timer	Interactive	Standard	Batch
Accept	500 ms	500 ms	5 s
TTFT after accept	5 s	60 s (may include load)	10 min
Inter-token stall	15 s	30 s	120 s
Wall clock	10 min	30 min	24 h

Any expiry → lease revoked, slot released (host told via `CANCEL`), job requeued front, host stat debited (stall counts hurt EWMA and reputation mildly; crash-looping hurts a lot).

5.5 Mid-stream failover

Gateway keeps the canonical emitted-token buffer per job. On host failure after first token: job re-enqueued with `resume_hint = {emitted_text}`; the new host receives the original prompt + emitted text as a continuation prompt (same sampling params, seeded RNG re-derived from job_id so temperature-0 flows are deterministic; for sampled flows the seam is accepted and marked in the response metadata `x-elephant-resumed: true`). Client stream continues seamlessly; the failed host's receipt is honored only up to the last gateway-acknowledged token batch.

5.6 Sizing reality

Phase-3 target (30B tok/day ≈ 350k tok/s aggregate, ~700 req/s at ~500-token mean completions): a scheduler shard handles one model's dispatch at <10k decisions/s trivially in memory; NATS JetStream at these rates is far below its millions-msgs/s envelope; gateways sized for ~10k concurrent SSE each → ~10–20 gateway pods. The control plane is small; the fleet is the datacenter.

---

6. Model lifecycle on the network

6.1 Catalog entry → servable build

Pipeline (ops-triggered, automated): pick base model (license-checked) → produce builds: `{AWQ-INT4, FP8, GGUF-Q4_K_M, GGUF-Q6_K}` as applicable → run eval gate (perplexity delta vs reference + task suite; quantization must stay within configured tolerance) → sign → publish to CDN + torrent → `catalog.json` update. Each build pins its engine image digest and launch args (I2: hosts run only these).

6.2 Prewarmer loop (per model, 30s tick)

```
demand_forecast = max(queue_depth_now,
                      EWMA_1h(arrival_rate) × p95_service_time × safety(1.3))
capacity_warm   = Σ slots_free(warm nodes)
if capacity_warm < demand_forecast:
    need = demand_forecast - capacity_warm
    candidates = cold-capable nodes (weights on disk, VRAM free, policy ACTIVE)
                 ranked by: disk-warm > net-fast, reputation, price
    send LOAD commands covering `need` slots (stagger 10s, cancel-on-overshoot)
if capacity_warm > 2× demand_forecast for 15 min:
    send UNLOAD to lowest-scoring surplus nodes (never below floor=2 nodes/model)
```

Load-time model (from cold-start research): NVMe→VRAM ≈ 12–15s for a 40GB Q4 70B at 3 GB/s + engine init 10–30s; disk-cold adds download at host's link speed (P2P-assisted, §7.3). The prewarmer treats `time_to_warm` per node as data (measured, per host) — not an assumption.

6.3 LoRA adapters

Adapters are first-class catalog artifacts (~10–200 MB): `{adapter_id, base_build, rank, sha256, eval_report}`. vLLM nodes preload the top-N popular adapters for their build (LRU in host RAM, paged to GPU on demand — S-LoRA/Punica pattern, +~2ms/token); a job's `adapters[]` is a routing constraint like residency. llama.cpp nodes: GGUF-merged adapter variants only (llama.cpp multi-LoRA hot-swap is not production-grade) — the registry auto-produces merged GGUF builds for the top adapters.

---

7. `mahout` — the host executable

7.1 Process model

Single Rust binary, ~15 MB static (musl on Linux). Crates: `mahout-core` (supervisor, state machine), `mahout-probe`, `mahout-bench`, `mahout-runtime` (engine adapters), `mahout-cache`, `mahout-net` (NATS/QUIC), `mahout-policy`, `mahout-update`. Runs as:

- **Linux**: systemd unit `mahout.service` (user or system scope); engines via rootless Podman (preferred) or Docker; GPU via CDI/nvidia-container-toolkit or ROCm devices.
- **Windows**: Windows Service; engines via WSL2-backed containers when available, else native llama.cpp (bundled, signed) — vLLM tier requires WSL2+CUDA (guided setup).
- **macOS**: launchd agent; native llama.cpp with Metal (no containers; the llama-server binary ships inside the signed mahout bundle).

Supervisor state machine per node:

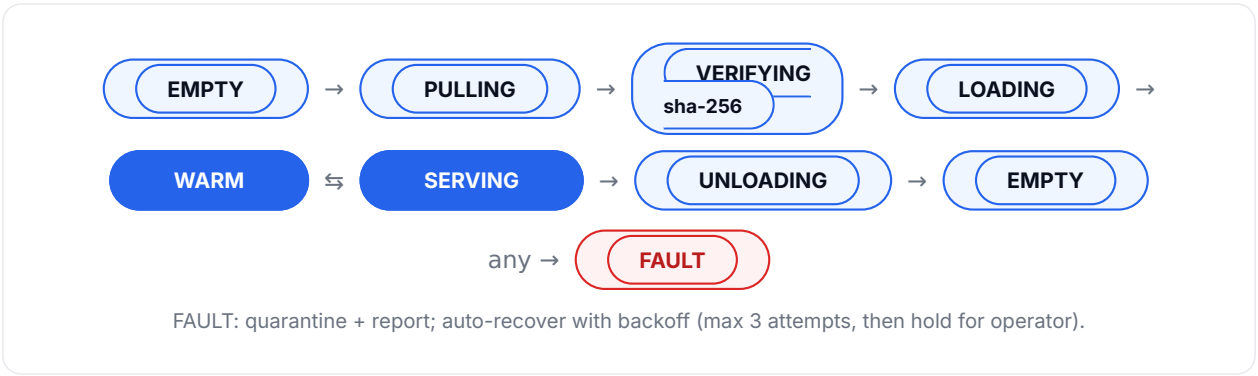

Figure 3 — Node lifecycle state machine (per engine instance, supervised by mahout).

7.2 Probe (on install, on hardware change, weekly)

Collected: GPU inventory via NVML / ROCm-SMI / Metal (`name, vram_total, pcie gen/lanes, driver, compute cap`), CPU (`model, cores, AVX512/AMX flags`), RAM, disks (free space + measured sequential read on the cache path — 1 GiB sample), network (download/upload measured against 3 relay PoPs, 10s each), NAT type (STUN-style via relay), OS/virtualization flags. Everything lands in the registry as claims pending bench confirmation. **No capability is schedulable from probe data alone.**

7.3 Cache manager

- Content-addressed store: `<cache_dir>/objects/<sha256>` with a manifest DB (SQLite); quota set by host (default 100 GB, slider in dashboard).
- Fetch strategy: BitTorrent-compatible swarm (weights are chunked; registry publishes infohash; our CDN seeds as webseed) → falls back to plain HTTPS. Upload participation is a host policy toggle (default on, capped at 50% of measured uplink); seeding earns a small bandwidth credit (metered by signed transfer summaries between peers, sampled-verified).
- Eviction: LRU by last-served, pinned models exempt (prewarmer can pin), never evict below the build currently loaded.

7.4 Bench (after probe; revalidated weekly + on driver change)

1. Registry selects the **reference build** for the hardware class (e.g. 24 GB NVIDIA → Qwen3-32B-AWQ on vLLM; 8 GB → Llama-3.1-8B-Q4 GGUF).
2. Fixed, versioned prompt set (32 prompts, mixed lengths) at fixed sampling (temp 0, seed 0), run at concurrency 1, then max stable concurrency (binary search until p95 inter-token > threshold or OOM).
3. Reported: `toks_per_sec@1` , `toks_per_sec@max` , `slots_max` , `ttft` , `load_time` , plus **TOPLOC commitments for every benchmark generation** — the verifier replays 4 of 32 on trusted hardware; activation-commitment mismatch = hardware/driver/precision anomaly → bench rejected.
4. Result defines which builds this node may advertise and its `slots_total` . A node can never advertise a model it hasn't successfully benched or served.

Anti-spoof properties: capabilities derive from measured tokens with verified commitments (a CPU pretending to be a 4090 cannot fake 400 tok/s of correct activations); re-benchmarks are unannounced (control-plane triggered); vitals cross-check (a "4090" drawing pattern-inconsistent timings gets flagged).

7.5 Runtime adapters

vLLM adapter (Linux/WSL2, NVIDIA ≥ Pascal 16GB / AMD RDNA3+ 20GB):

- Launches the pinned image: `podman run --rm --device nvidia.com/gpu=N --network=slirp4netns:allow_host_loopback=false --read-only --tmpfs /tmp -v cache:/weights:ro <image@digest> --model /weights/<build> --max-num-seqs <slots> --enable-lora ...` — args come from the build manifest verbatim; mahout only fills mount paths and ports.
- Container network egress: **none** (engine talks only to mahout via a localhost port bound to the container). Weights mounted read-only. This is the I2/I3 enforcement point on the host side.
- Health: `/health` poll 5s; `/metrics` scraped for slot/KV occupancy (feeds heartbeat).
- TOPLOC: engine image includes the activation-commitment hook (top-k hidden-state LSH per 32 tokens, per the TOPLOC construction); commitments surface in the completion response extension and pass through to `JOB_DONE`.

llama.cpp adapter (everything else):

- Bundled `llama-server` (signed, per-platform builds with CUDA/Metal/Vulkan/ROCm variants), launched with build-manifest args (`-ngl`, ctx, batch); OS-level sandbox: seccomp + landlock on Linux, App Sandbox on macOS, AppContainer on Windows; no network except localhost.
- TOPLOC hooks are not upstream in llama.cpp: these nodes run in **replication-verified mode** instead — higher sampling rate (5% vs 1%), logprob fingerprints recorded (top-4 logprobs every 16th token) for statistical cross-checking, and they are excluded from the attested/sovereign tier. (Porting the commitment hook to llama.cpp is a roadmap item that upgrades the whole consumer fleet's trust level.)

7.6 Policy engine (host controls; all local, reflected to registry)

`availability windows` (cron-like), `pause on user activity` (input idle < 5 min) and `pause on fullscreen/game` (default on for desktop installs), `max power` (enforced via `nvidia-smi -pl` / ROCm equivalent), `vram_reserve`, `price_floor` (€/Mtok multiplier ≥ configured floor), `disk quota`, `seeding on/off + cap`, big red **pause now**. Pausing mid-job: node drains (finishes in-flight, max 60s, else jobs fail over) — drain compliance is tracked and rewarded (clean drains don't hurt reputation; yank-the-cord does, mildly).

7.7 Updater

TUF-style: root/targets/snapshot/timestamp keys; binary diffs; staged rollout rings (canary 1% → 10% → 100%, gated on fleet health metrics); mandatory floor version enforced by control plane (nodes below floor get no leases). Engine images and weights are versioned independently of the binary (registry-driven), so most "updates" don't touch mahout itself.

7.8 Local observability

`mahout status` TUI + local web panel (`localhost:4200`): current jobs (counts only, never content — I3), earnings live (from signed receipts, before control-plane settlement), temps/power, reputation, logs. Earnings shown here are provisional (receipt-based) and reconcile with the monthly statement.

---

8. Verification, receipts, reputation

8.1 Receipt (CBOR, signed by host key)

```
{ v:1, job_id, host_id, node_id, model_build, adapters[],
  prompt_tokens, completion_tokens, tier,
  toploc: [ {range:[i,j], commitment: 258B} ... ],     // vLLM nodes
  logprob_fp: bytes?,                                   // llama.cpp nodes
  timing: {accepted_at, ttft_ms, done_at},
  gateway_id, price_version, sig }
```

`receiptd` accepts iff: signature valid, job exists, gateway's independent count within 1%, host not quarantined. Accepted → billable → Merkle-anchored (§3.7).

8.2 Verification pipeline

- **Sampling**: 1% of vLLM-node jobs, 5% of llama.cpp-node jobs, 100% of a host's first 50 jobs, 100% of disputed jobs, weighted toward high-value receipts. Selection is deterministic-after-the-fact: `verify if H(job_id || daily_beacon) < threshold` where `daily_beacon` is published after the day closes (host cannot know at serving time which jobs will be checked; beacon = hash of the day's BSV anchor txids — the chain provides the unpredictable randomness for free).
- **Replay**: verifier re-runs the prompt on anchor-fleet hardware with the same build/params. vLLM path: recompute activation commitments, compare (TOPLOC tolerances absorb GPU nondeterminism; wrong model/quant/prompt = deterministic mismatch). llama.cpp path: compare logprob fingerprints statistically (KS-test over top-4 logprob agreement; threshold calibrated per build) + exact-output comparison for temp-0 jobs.
- **Verdicts**: `ok` | `anomaly` (one-off, recheck ×3) | `fraud` (systematic). Fraud → epoch payout withheld, node quarantined, account review; second offense → ban + forfeiture of pending balance (contractually specified). All verdicts are receipt-linked and appealable (we hold the evidence: commitments + replay).

8.3 Reputation (per node, [0,1])

```
rep = 0.35·completion_rate_30d          // jobs done / jobs accepted
    + 0.20·verification_pass_rate       // Bayesian, Beta(α,β) prior Beta(2,1)
    + 0.15·latency_consistency          // 1 - cv(inter-token intervals)
    + 0.15·availability_adherence       // uptime within self-declared windows
    + 0.15·tenure_factor                // saturates at 60 days
```

Tier floors: interactive ≥ 0.65, sovereign ≥ 0.80 + attestation/contract, batch ≥ 0 (that's the ramp). Reputation is per-node, non-transferable, decays toward the prior when idle (30-day half-life) — a sybil farm gains nothing by cycling identities (each restart re-enters the 50-job full-verification ramp earning batch rates only).

9. Training jobs on the same fleet (forward-compatibility)

The job protocol §4.3–4.4 deliberately fits non-inference work: a `JobLease` with `kind: TRAIN_LORA` carries a dataset shard URI (content-addressed, downloaded via §7.3) instead of a prompt; the "stream" returns loss curves; the receipt commits to the produced adapter's sha256. Verification = held-out-loss check on the anchor fleet (Gauntlet-style measured-contribution scoring). RL rollout generation (`kind: ROLLOUT`) is literally inference with a task-spec prompt + TOPLOC receipt — zero protocol changes. This is how future training-shaped workloads ride the same rails without a second network.

10. Data model (Postgres, abridged)

```
accounts(id, kind client|host|both, email, kyc_level, created_at, …)
api_keys(id, account_id, hash, limits jsonb, residency_default, status)
hosts(host_id pk, account_id, pubkey, cert_serial, region, status, enrolled_at)
nodes(node_id pk, host_id, gpu_inventory jsonb, bench jsonb, engine, status)
models(model_id pk, family, params_b, license, aliases text[])
builds(build_sha pk, model_id, quant, engine_image_sha, size_bytes,
       eval_report jsonb, uris text[], status draft|live|retired)
adapters(adapter_id pk, base_build, rank, sha256, eval_report jsonb, status)
price_book(version pk, model_id, tier, residency, eur_per_mtok_in, _out,
           host_share numeric, valid_from)
jobs(job_id pk, account_id, model_id, build_sha, node_id, tier, residency,
     state, price_version, enqueued_at, done_at)          -- partitioned by day
receipts(receipt_id pk, job_id, host_id, payload bytea, sig, gateway_counts jsonb,
         status accepted|mismatch|fraud, anchor_id)        -- partitioned by day
anchors(anchor_id pk, merkle_root, txid, bump bytea, anchored_at)
ledger(entry_id pk, account_id, kind debit|credit|payout|adjustment,
       amount_eur numeric, job_id?, epoch, created_at)     -- append-only
reputation_events(node_id, kind, delta, receipt_id?, created_at)
verification_events(receipt_id, verdict, evidence jsonb, verifier_node, created_at)
```

ClickHouse: `usage_events` , `heartbeats` , `latency_samples` (TTLs 90 days). Content data (prompts/completions): **no table exists — by design** (I3).

11. Security threat model (condensed)

Threat	Vector	Mitigation
Malicious host reads prompts	RAM inspection on own machine	acknowledged residual for standard tier (ToS-disclosed); sovereign/ attested tier = TEE or contracted hosts only; routing constraint enforced at scheduler
Malicious host fakes work	wrong/smaller model, garbage	TOPLOC commitments + unpredictable post-hoc sampling (§8.2), payout withholding
Sybil host farms	many fake identities	measured bench + verified commitments, per-node ramp, batch-only start, no transferable reputation
Malicious client attacks host	prompt as exploit	prompts are data to a sandboxed engine; no tool execution host-side, ever; engine containers: no egress, read-only FS, resource-limited
Us (or compromise of us) pushing malware	update channel	open-source binary, TUF multi-key signing, reproducible builds (goal), staged rollout, engine-image allowlist baked into release
Relay compromise	traffic snooping	inner TLS end-to-end gateway↔host; relays see ciphertext + metadata only
Control-plane DB breach	billing PII	no content data exists (I3); keys hashed; payout details tokenized at PSP
Receipt forgery / repudiation	host or platform lies about work	mutual: host signature + gateway independent count + public Merkle/BSV anchor
Model weight exfiltration	host copies weights	open-weight models only — weights are public by definition; licenses tracked per build; gated/proprietary models are explicitly out of scope for the community fleet (anchor fleet only, if ever)

12. Phase 0 MVP cut (what we actually build first)

In: gateway (chat/completions + streaming, API keys, prepaid balances), single-region control plane, scheduler (one shard, no batch tier), registry with the consumer/workstation slice of the launch catalog × 2 builds each (Llama-3.1-8B, Qwen3.8-27B, Gemma-4-26B-A4B, Llama-3.3-70B, gpt-oss-120b; AWQ + GGUF — datacenter MoE models DeepSeek V4-Flash / MiniMax M3 / GLM-5.2 / Qwen3.5-397B serve from the anchor fleet with vLLM-native quants), **mahout Linux for NVIDIA and AMD — both day one**: vLLM adapter with CUDA and ROCm engine images (ROCm ≥6.3 covers RDNA3 consumer cards incl. RX 7900 XT/XTX; separate pinned image per backend, same manifest contract), llama.cpp adapter with CUDA/ROCm/**Vulkan** builds (Vulkan is the AMD fallback when ROCm isn't installable), probe via NVML + ROCm-SMI, per-backend reference bench builds (24GB NVIDIA → Qwen3-32B-AWQ/vLLM-CUDA; 24GB AMD → Qwen3-32B-AWQ/vLLM-ROCm, llama.cpp-Vulkan fallback); relays ×2, receipts + signatures + hourly ledger (BSV anchoring stubbed to a log), replication-verification only (TOPLOC hook lands in Phase 1), ~20 known-friendly hosts + anchor fleet (which itself includes RX 7900 XTX cards — AMD is exercised from the first internal deployment, not a port). **Out** (Phase 1+): Windows/

macOS, TOPLOC, batch tier, prewarmer (manual pinning instead), LoRA serving, auto-router, hole-punched direct paths (relay-only first), P2P weight distribution (CDN-only first), payout automation (manual SEPA batch).

Build estimate: ~2 engineers × 3–4 months to Phase 0 exit with the cut above; the long poles are mahout's engine-adapter hardening and the failover-correct gateway streaming path.

Companion doc: WHITEPAPER.md (market, architecture, economics).