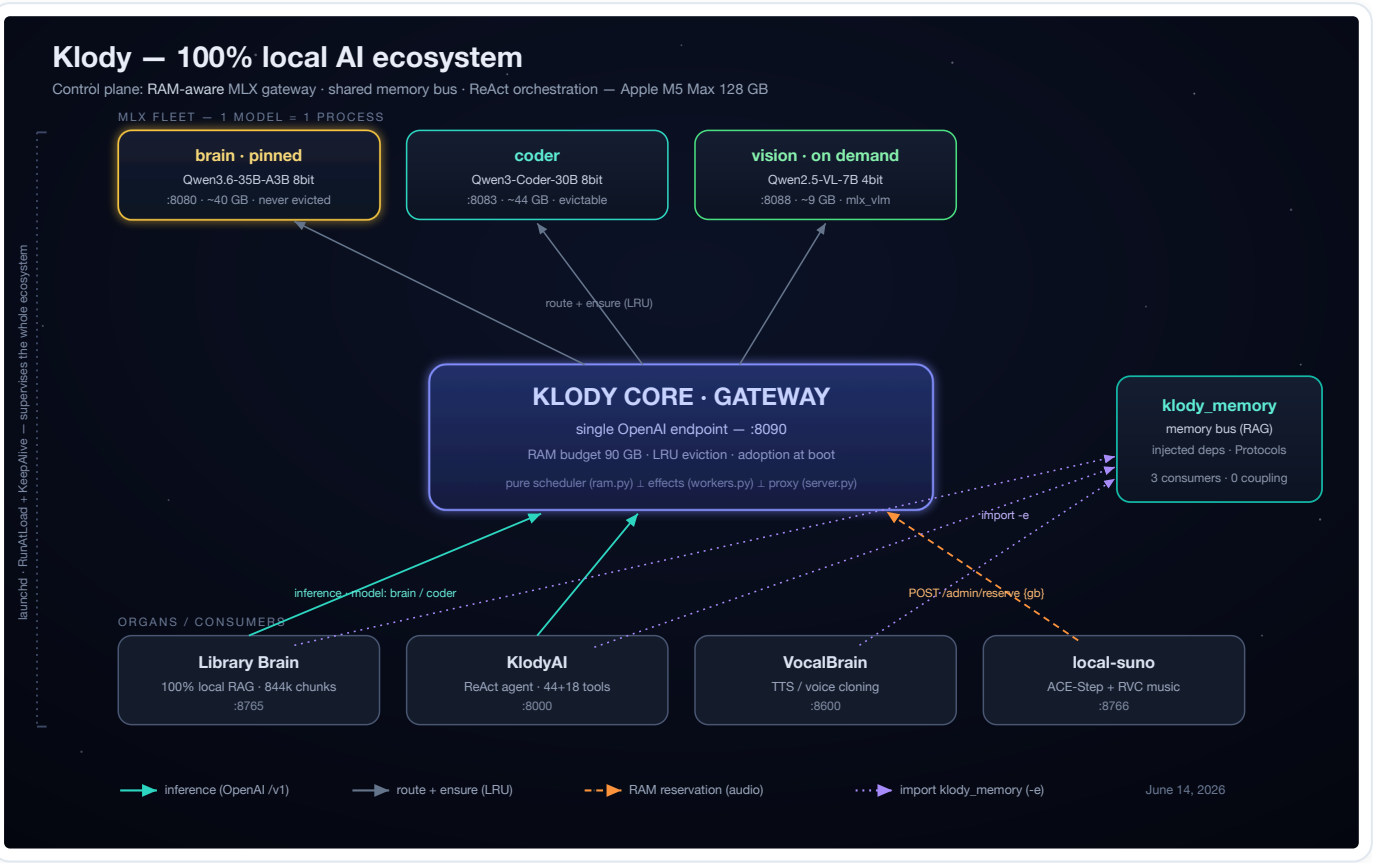
Klody — Architecture of a 100% Local AI Ecosystem

> **Technical portfolio** — design, methods, practices and stack. Author: **klodynlov** · Machine: Apple **M5 Max, 128 GB** · macOS (arm64) · 100% local, zero cloud, zero telemetry. Period covered: **May 9 → June 14, 2026** · Document generated **June 14, 2026**.



0. Executive summary

In five weeks I designed and put into production a **fully local AI ecosystem**: five specialized applications (document RAG, coding agent, speech synthesis, music generation, desktop dashboard) and then — most importantly — the **control plane** that unifies them: **Klody Core**.

The core of the work is not "yet another AI app" but **infrastructure**: a shared, RAM-aware inference *gateway* that pools multi-gigabyte models across every application under a single memory budget with smart eviction. Around it sits a reusable **memory bus** (the RAG engine extracted into a dependency-injected package) and an **orchestrator agent** (a ReAct loop) that drives the organs.

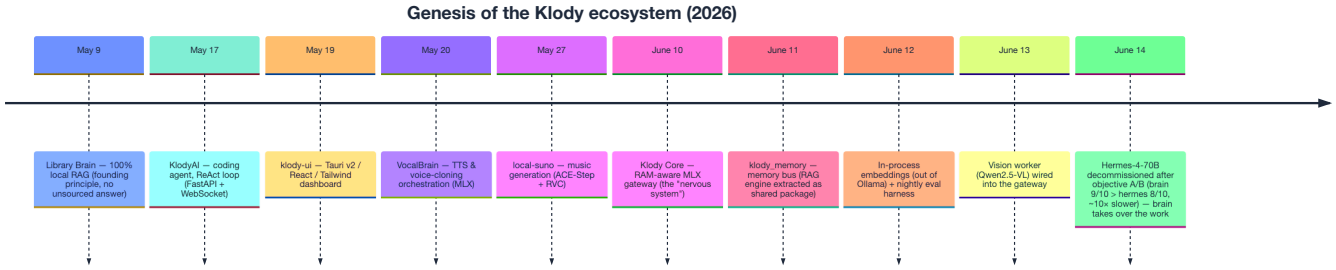
What this project demonstrates:

Skill	Concrete evidence in the code
Systems architecture	Spotted a cross-cutting gap ("5 organs, no nervous system") → built a *control plane*, not a 6th feature.
Concurrency engineering	Async scheduler, lock + double-check, in-flight request tracking, cleanup guaranteed via `BackgroundTask` (zero state leaks).
Security posture	Enforced loopback bind, anti-OOM body cap, anti-DoS guards (`nan` / `inf`), telemetry off — threat-modeling a local service.
Disciplined refactoring	Extracted an 11-module package via dependency injection + back-compat *shims*, validated by ~1,100 tests + a quality *gate* after each sub-lot (*strangler fig* pattern).
ML / LLMOps quality	Explicit distinction "the code works" (unit tests) vs "the models still answer well" (nightly eval harness, pinned baselines, regression detection).
Measurement discipline	RAM measured with `vmmap` (not `ps rss` — deep knowledge of Metal unified memory); embedding migration validated by `cos = 1.0000` **before** avoiding a re-index of 844,000 vectors.
Operational maturity	Everything is a `launchd` service (RunAtLoad + KeepAlive), dedicated venvs to break circular deps, health endpoints, structured logs, eval cron.

Scale: ~**381 commits** across 6 repos, ~**2,350 tests** green, ~**4,000 lines** for Klody Core alone (gateway + memory package).

1. Genesis — the timeline

The ecosystem was not designed all at once: it **emerged**. Five standalone applications first, then the realization that they shared a common gap, and finally the infrastructure that federates them.



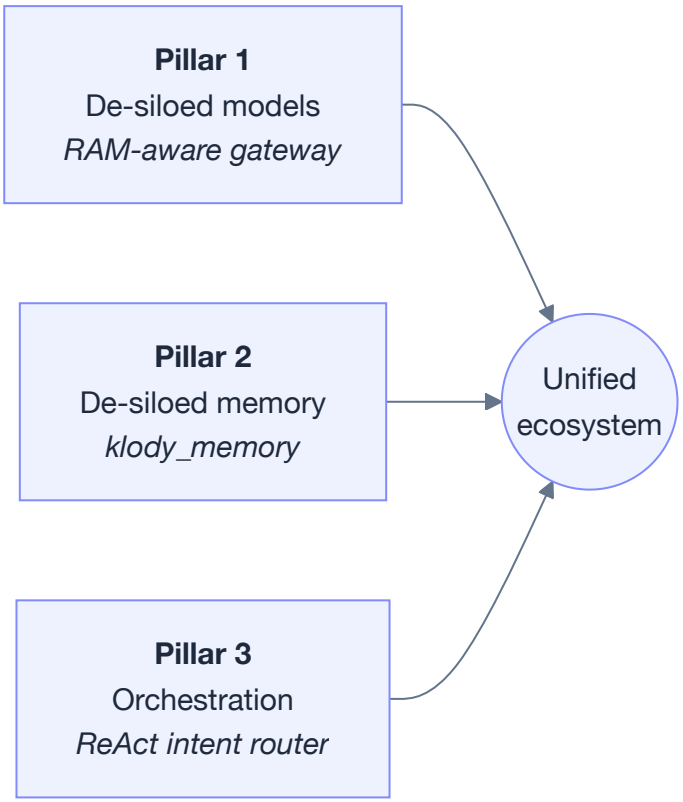
The architectural turning point (June 10)

The founding observation, stated then validated: *the five projects are excellent organs, but with no nervous system.* Each loaded its own models, managed its own memory, exposed its own service on its own port. The measured consequence: **~208 GB of redundant weights on disk** and **N duplicated inferences** in RAM.

The decisive decision — and the mark of architect-level thinking — was to **not rewrite the organs but to *front* them**. Klody Core is a personal *control plane*, owned as such ("hardcoded for ~5 services, not a framework"). The effect is multiplicative: *every hour invested in the Core increases the value of everything that already exists.*

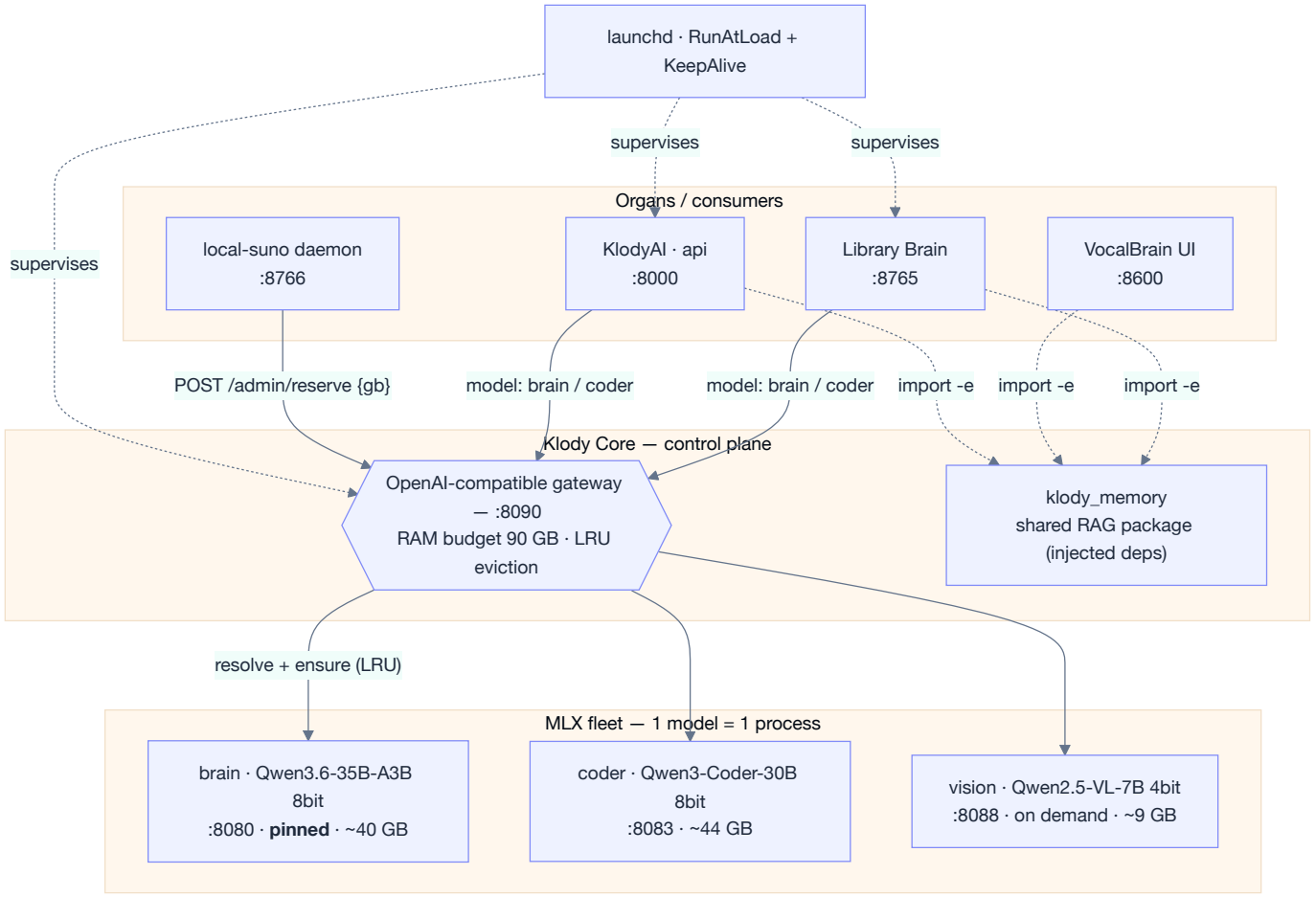
2. Overview — the "nervous system" thesis

Klody Core rests on **three pillars**, tackled in order and all shipped:



1. **The gateway** — a single, shared, RAM-aware inference endpoint.
2. **The memory bus** — the RAG engine promoted to a reusable package.
3. **The intent router** — the ReAct agent that drives the organs (speak, search the books, compose, remember).

Runtime topology



Two distinct flows coexist:

- **Inference flow** (solid line): organs speak standard OpenAI to the gateway, which resolves the alias, guarantees the model is resident, and relays (streaming included).
- **Memory flow** (dotted): three applications **import** the same `klody_memory` package (`pip install -e`) — no HTTP bridge, shared code identity.
- **Reservation flow**: an audio job (ACE-Step/RVC) asks the gateway to **free RAM** before running.

---

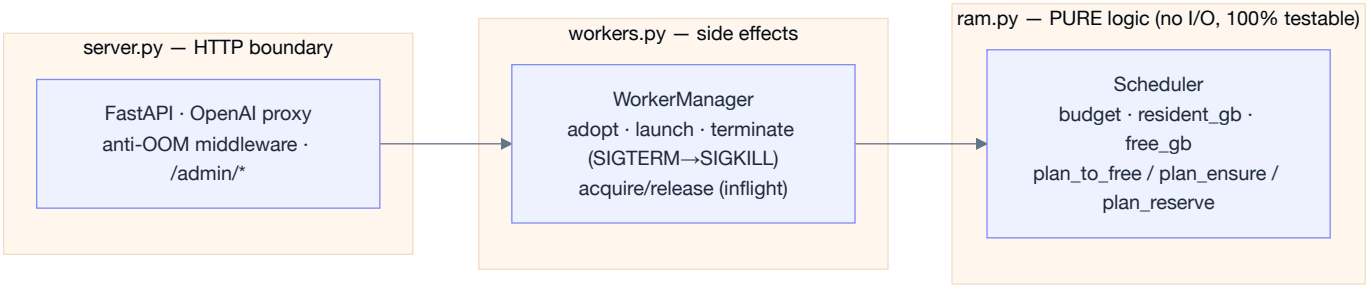
3. Pillar 1 — the RAM-aware MLX gateway

The problem, precisely

`mlx_lm.server` (the Apple Silicon inference stack) serves **a single fixed model per process**: no native hot-swap. Before the Core, each application launched/pinned its own server → the `brain` (~40 GB) and the `coder` (~44 GB) coexisted twice over, plus Ollama. On a 128 GB machine, that's the wall.

The solution: separate decision from effect

This is the project's single most important design choice, and it is textbook: **the pure planning logic is isolated from all I/O.**



- `ram.py` holds **no** subprocess, no socket: just decisions ("which workers to evict to fit `X` GB?"). It is fully testable offline.
- `workers.py` applies those plans to reality (launch/kill processes, track in-flight requests).
- `server.py` is the HTTP boundary (OpenAI proxy + admin).

This **hexagonal** separation is what distinguishes a throwaway service from a maintainable one.

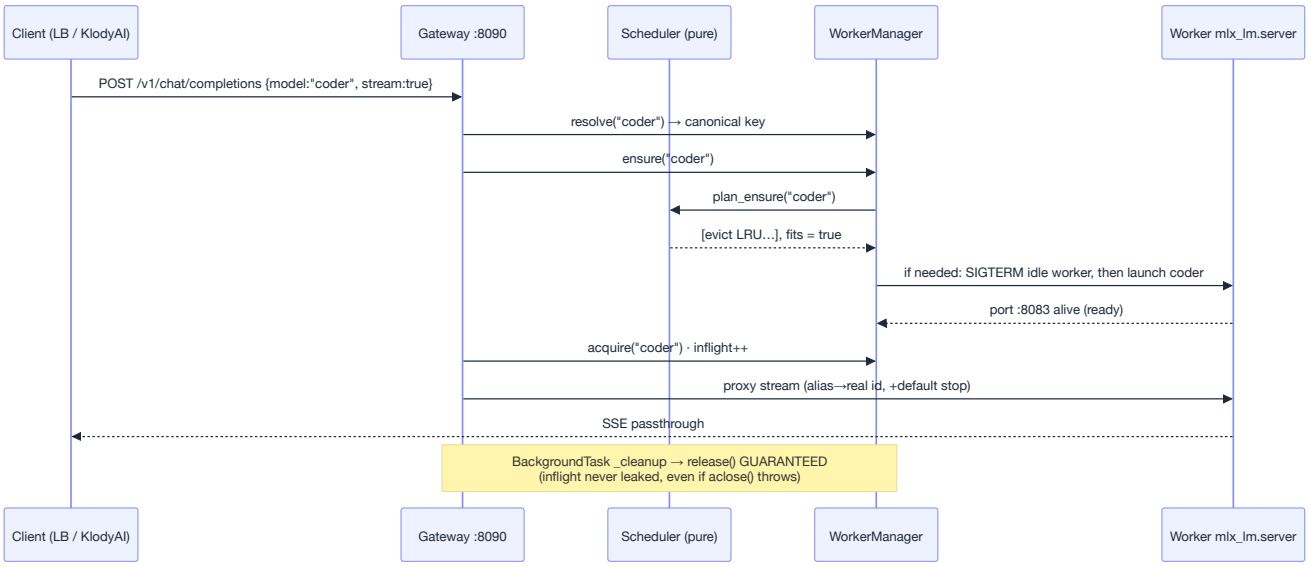
Key mechanisms

Budget & LRU eviction. A **90 GB** budget out of 128 (headroom for the OS + audio jobs). The `brain` is `pinned` (shared by Library Brain *and* KlodyAI → never evicted); other workers are evicted **least-recently-used first**, never a `pinned` worker nor one serving a request (`inflight > 0`).

Adoption. At startup the gateway **detects** workers already alive on their ports and *adopts* them instead of relaunching — restarting the gateway does not kill in-flight inference.

Audio reservation. `POST /admin/reserve {"gb": 24}` frees RAM *before* ACE-Step/RVC starts. The returned report reflects the **actually free RAM** (`satisfied`), not the plan — the caller must verify before launching its job.

Sequence of a (streaming) request



Details that reveal the level

These choices are not invented without scars — they are the signature of an experienced engineer:

- **No `inflight` leak in streaming.** `release()` runs via a Starlette `BackgroundTask`, executed on *every* path (normal end, client disconnect, cancellation). Without it, if `aclose()` threw on a broken transport, the `coder` would stay `inflight > 0` **forever** → permanently non-evictable, RAM pinned for good.
- **Honest RAM accounting.** If a worker survives `SIGTERM` *then* `SIGKILL`, `_terminate` **restores the real state and throws** instead of pretending it freed the RAM — lying here would cause an OOM on the next audio job.
- **Race closed early.** `running = False` is set **before** the first `await` of `terminate`, under the lock, to close the fast path of `ensure()` during shutdown.
- **Raw relay in non-streaming.** We do not re-parse the worker's response (`r.json()`): a worker in error can return HTML/text/empty → a re-parse would throw and hide the real status behind an opaque 500.
- **`vmmap`, not `ps rss`.** MLX weights live in Metal buffers (unified memory) → `ps rss` shows ~400 MB for a 40 GB model. The footprint is measured with `vmmap -summary` (Physical footprint).

Extensibility, proven

The registry (`config.py`) is purely declarative. Two recent extensions validated it without touching the core:

- **Hermes-4-70B** (`:8091`, *since decommissioned*): an on-demand worker that **proved the registry's extensibility** before being retired. A non-trivial pitfall was solved along the way — the repo's `eos_token_id` (`128001`) did not cover the end-of-turn `<|eot_id|>` (`128009`) → *runaway* generation; the robust fix is kept in the gateway: injecting `stop: ["<|eot_id|>"]` by default (survives a re-download of the HF cache). **Retired on June 14** after an objective A/B (`brain` 9/10 vs `hermes` 8/10 for ~10× the latency): no gain, `brain` takes over the work. *Knowing how to remove a component that doesn't earn its keep is an architect's decision too.*
- **Vision worker** (`:8088`, Qwen2.5-VL): launched via `mlx_vlm.server` (not `mlx_lm.server`) — the launcher adapts its arguments by module and tolerates a multi-GB download on first launch.

---

4. Pillar 2 — the `klody_memory` memory bus

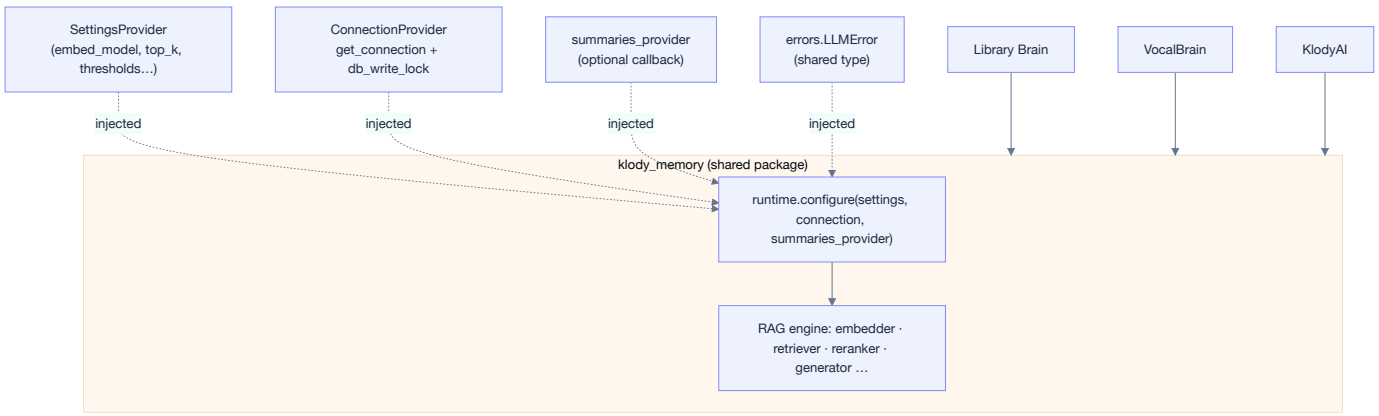
The idea

The most polished RAG engine lived inside Library Brain. Rather than copying it into each app (guaranteed drift) or exposing it over HTTP (network coupling), I **extracted it into a reusable, dependency-injected package** — the source of truth stays single, consumers import it.

Method: *strangler fig* + dependency injection

The extraction (11 modules: `models`, `embedder`, `retriever`, `reranker`, `generator`, `prompt_builder`, `scorer`, `translator`, `llm_provider`, `repo_retriever`, `web_search`) followed a boundary defined *before* moving a single line, then sub-lot by sub-lot, with the **~1,100-test** suite and a **RAG quality gate** replayed after each step.

The *seam* is four injected dependencies via Python `Protocol`s:



Key decision: the package **does not own the database** — it receives `get_connection` + `db_write_lock`. This settles the SQLite "second writer" problem outright: each consumer keeps its own database, its own lock, its own settings.

Back-compat: Library Brain's old `rag/*.py` paths become **shims** that re-export from `klody_memory` — zero breakage, code identity preserved.

Genericity proven by 3 consumers

Consumer	Dedicated DB	Domain	Imports Library Brain?
Library Brain	`library_brain.db` (844k chunks)	books	— (source)
VocalBrain	`~/.vocalbrain/memory.db`	audio ("sad piano track" → the right ballad)	**none**
KlodyAI	`logs/semantic_memory.db` (1,533 memories)	agent sessions	**none**

That VocalBrain semantically recalls *the right track* with **zero import** of Library Brain is the **empirical proof** that the abstraction is correct.

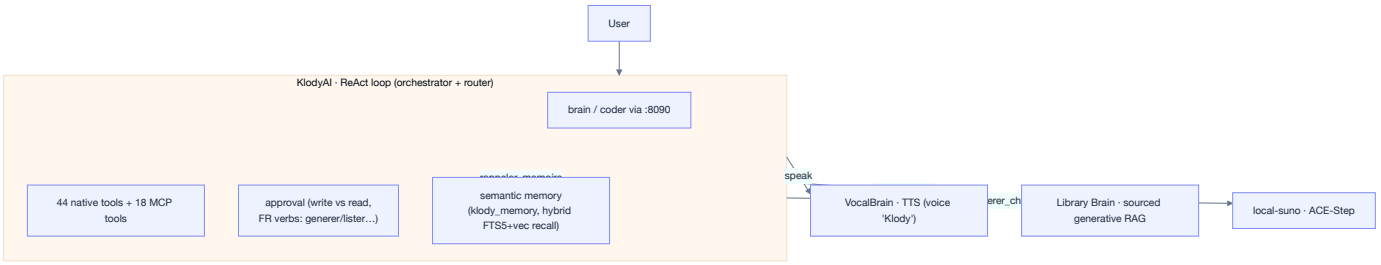
A migration decision validated by measurement

Embeddings went through Ollama (a separate daemon). Before migrating to in-process `sentence-transformers`, I **measured** compatibility on the real corpus (`measure_embed_compat.py`): `cos(emb_ollama, emb_st) = 1.0000` everywhere → **zero re-encoding of the 844,000 existing vectors**. A risky migration turned into a trivial config change, *because* it was measured first.

---

5. Pillar 3 — the intent router (ReAct agent)

The gateway de-silos the **models**, `klody_memory` de-silos the **memory** — but an **orchestrator** was missing. That's **KlodyAI**, the coding agent, promoted to router: its **ReAct** loop (Reason + Act) selects and calls tools, including the "motor arms" that actuate the other organs.



Notable points:

- **Reuse discovery**: when building the "arms", 2 of the 3 already existed (the RAG bridge and the music bridge via MCP). The real work focused on the missing tool — `speak` — and on **approval safety** (the French verbs `generer/entrainer/supprimer…` were classified as *write* and passed without validation).
- **TTS pitfalls solved empirically**: a multi-sentence text in one block → the EOS never came (163 s of saturated WAV); fix = segmentation by `\n` (the model's split pattern) + full language names (`french` , not `fr`).
- **In-process semantic memory** (bge-m3, `sentence-transformers`), automatic mirroring of facts/sessions, **1,533 memories** backfilled, `rappeler_memoire` natural-language tool.

---

6. The organs

Application	Role	Stack	Status
Library Brain	100% local RAG — answers *only* from the books (founding principle)	FastAPI · SQLite FTS5 + sqlite-vec · bge-m3 · ~5,800 books / 844k chunks	In prod (quality gate green)
KlodyAI (`klody-code-ai` + `klody-ui`)	ReAct coding agent, desktop dashboard	FastAPI · WebSocket · Tauri v2 · React · Tailwind v4	In prod (1,132 tests)
VocalBrain	TTS & voice-cloning orchestration	Python · Pydantic v2 · mlx-audio · SQLite · 4-D model router	In prod (38 tests)
local-suno	Music generation + voice conversion	ACE-Step · RVC · daemon	In prod (37 tests)
RetroEcho	Local-AI macOS app (pure Swift)	Swift · MLX (LoRA)	Phases 1–8
GrainForge	Embedded generative groovebox	Teensy 4.1 + Pi 4 + M5 Max	Phase 0 (hardware)

> An owned and upheld architectural constraint: **RetroEcho stays pure Swift** — it consumes the gateway *optionally*, never via a Python bridge at runtime. Knowing where *not* to extend an abstraction is also a sign of maturity.

---

7. Operations (Ops)

`launchd` service map

Everything runs as a supervised service (RunAtLoad + KeepAlive: starts at login, restarts on crash).

`launchd` service	Port	Role
`com.klody.core-gateway`	8090	MLX gateway
`com.klody.api`	8000	KlodyAI backend
`com.librarybrain.server`	8765	Library Brain web
`com.klody.vocalbrain-ui`	8600	VocalBrain UI
`com.klody.localsuno-daemon`	8766	music-generation daemon
`com.klody.klody-mcp` / `web-mcp` / `gmail-mcp` / `vocalbrain-mcp`	8087 …	MCP servers (agent tools)
`com.klody.nightly-eval`	—	quality eval, **03:30** every night

Engineering details: a **dedicated** venv `~/klody-core-env` to **break the circular dependency** (the gateway no longer depends on Library Brain's venv); the `launchd` `PATH` must include `/usr/sbin` (otherwise `lsof` is unavailable → degraded PID detection) — a pitfall diagnosed and fixed.

The nightly eval harness — the maturity piece

> Unit tests prove **the code** works. They do **not** prove **the models** still answer well. This distinction, rare even on professional teams, is explicitly addressed.

`nightly_eval.py` (`launchd` cron at 03:30) in three independent sections:

1. **Gateway health** — `/admin/status` + latency of a `brain` completion.
2. **Full RAG gate** for Library Brain — `--check --generate` (retrieval **and** generation vs pinned baseline): keyword coverage, refusal rate, **zero hallucination**.
3. **Code pass@1** — *golden* prompts → `coder` → the generated code is **executed under pytest** (regression if < 2/3).

A dated Markdown report, **macOS notification only on regression** (silence = all good), exit code 1 if red.

Telling anecdote (false positive on June 13). The first automatic run went red (RAG gate collapsed). Diagnosis: *not a regression* — at night, `brain` (40) + `coder` (44) saturate the 90 GB; cold, loading embeddings caused **swapping** (brain ~13× slower). Warm the next morning: 100% green, same config. **Fix**: unload the `coder` (`/admin/unload`) *before* the nightly RAG gate — it only serves pass@1. Telling a regression from an environment artifact is experience.

---

8. Engineering methods & practices

A catalog of practices actually applied, observable in the code and Git history:

- **Pure-logic / side-effect separation** (hexagonal architecture) → maximum testability.

- **Dependency injection via `Protocol`** → decoupled package, 3 heterogeneous consumers.
- **Strangler fig** for package extraction → back-compat shims, no big bang.
- **Test-as-you-go**: full suite + quality gate **after each sub-lot**, never only at the end.
- **Adversarial pre-commit review** (multi-agent workflow) before the gateway's first commit: **10 confirmed issues out of 22 candidates** (1 critical, 1 major, 4 medium, 4 minor), all tracked and fixed (inflight leak, lying RAM accounting, `running` race, opaque 500s…).
- **Measurement-driven decisions**: `vmmap` for RAM, `cos = 1.0` before the embedding switch, tok/s measured before adopting a model.
- **Honest accounting / *fail loud***: never claim an unverified state (freed RAM, finished request).
- **Security by default**: enforced loopback (refuses a non-local bind without explicit opt-in), anti-OOM cap, `nan` / `inf` anti-DoS guards, HF/MLX telemetry off.
- **Idempotence**: backfill / bootstrap scripts re-runnable without damage.
- **Documenting the *why***: READMEs and comments explain trade-offs and edge cases, not the trivial *what*. The project notes serve as living **ADRs**.
- **Git hygiene**: Conventional Commits, protected `main` branch (PR + signed squash), private GitHub backups.

---

9. Architecture decisions (ADR — excerpts)

> Reformulated in ADR form from the real history.

ADR-1 — Front, don't rewrite. *Context*: 5 organs, ~208 GB of redundant weights. *Decision*: a control plane that *adopts* existing services. *Consequence*: multiplicative value of the existing code; the Core stays small and focused.

ADR-2 — Pure scheduling logic, isolated from I/O. *Decision*: a `Scheduler` with no subprocess or socket; a `WorkerManager` applies the plans. *Consequence*: an offline-testable core (`test_ram.py`), lifecycle tested separately in hermetic fashion (mocks).

ADR-3 — The memory package does not own the database. *Decision*: inject `get_connection` + `db_write_lock`. *Consequence*: no "second writer" conflict, each consumer keeps its own DB/settings.

ADR-4 — In-process embeddings rather than a daemon. *Decision*: `sentence-transformers` loaded once per process (validated `cos = 1.0`). *Consequence*: one fewer daemon on the critical path; ~10 ms warm encoding; zero re-indexing.

ADR-5 — Distinguish "the code works" from "the model answers well". *Decision*: a nightly eval harness with pinned baselines, separate from unit tests. *Consequence*: detection of *quality* regressions (hallucination, coverage) invisible to classic tests.

---

10. Development-level assessment

> An **evidence-backed** self-assessment, written for the reader (recruiter, CTO, peer).

The body of work places the author at a **senior / staff software engineer** level, with a **specialization in local-AI infrastructure (Apple Silicon LLMOps)**. The signals, mapped to recognizable competencies:

Observed signal	Demonstrated skill	Level
Recognizing a cross-cutting gap and building a *control plane* rather than a feature	Systems thinking, architectural vision	Staff
Pure scheduler separated from effects (hexagonal)	Design for testability	Senior+
`inflight` leak, `running` race, `BackgroundTask` cleanup	Correct async concurrency	Senior+
Enforced loopback, anti-OOM, anti-DoS guards, telemetry off	Threat-modeling, security by default	Senior
Package extraction via DI + Protocols + shims, validated by 1,100 tests	Large-scale refactoring without breakage	Senior+
Nightly eval harness, pinned baselines, executed pass@1	ML / LLMOps quality	Rare (Senior/Staff ML)
`vmmap` vs `ps rss`, `cos = 1.0` before migration	Measurement discipline, systems depth	Senior
Diagnosing the nightly false positive (swap, not regression)	Production systems debugging	Senior+
`launchd`, dedicated venvs, health endpoints, logs, cron	Operational / SRE maturity	Senior
Knowing where *not* to extend (RetroEcho stays pure Swift)	Architectural judgment	Senior+

One-sentence summary for a CV / job description:

> *An engineer able to design and operate a resource-aware, multi-model LLM inference infrastructure, extract reusable abstractions without breaking what exists, and institute quality assurance specific to AI systems — all in a 100% local, memory-constrained, production environment.*

Substantiated areas of expertise: software architecture · distributed systems (at single-machine scale) · LLMOps / MLX inference · RAG (retrieval, reranking, cross-lingual embeddings) · agents (ReAct, tools, MCP) · asyncio concurrency · application security · macOS DevOps (`launchd`) · audio DSP (TTS, voice cloning) · desktop (Tauri/React) · C++/JUCE (adjacent audio projects).

---

11. Project metrics

Repo	First commit	Commits	Tests	Role
`library-brain`	May 9	150	~1,113	local RAG
`klody-code-ai`	May 17	137	1,132	ReAct agent
`klody-ui`	May 19	31	—	Tauri dashboard
`vocalbrain`	May 20	23	38	TTS / voice
`local-suno`	May 27	22	37	music
`klody-core`	June 10	18	28	control plane
Total	—	**~381**	**~2,348**	—

Other figures: Klody Core ≈ **4,000 lines** (gateway ~750 + `klody_memory` ~3,300) · a **3-model fleet** (35B/30B/7B-VL) · **844,738 chunks** indexed · **~208 GB** of redundancy removed through pooling · `brain` 36.7 GB measured / `coder` 42.7 GB (peak 62.3).

12. Growth areas / owned limitations

> Naming them is deliberate: a credible portfolio also shows clear-sightedness about its own limits.

- **Single-machine, no horizontal scale** — by design (a *personal* control plane), but the gateway is not multi-node.
- **Gateway without authentication** — mitigated by the loopback bind; network use would require an auth layer (the groundwork is laid: non-local bind refused without opt-in).
- **Static `est_ram_gb`** — measured but frozen estimates; continuous measurement (`footprint`) would refine eviction.
- **RAM tension settled by the A/B** — `brain` + `coder` + `hermes` (40+44+44) together exceeded 90 GB; the harness's objective A/B settled the dilemma by **removing** `hermes` (zero quality gain for ~10× the cost). The `brain` + `coder` + `vision` fleet now fits under budget — the tension is resolved, not merely worked around.
- **Tracked minor debts**: the package `pyproject` at `dependencies=[]` (owned, later phase); a few non-thread-safe global memoizations (noted in the audit); the FTS `chunks_fts` index to rebuild (`rebuild`).
- **Business coupling of the RAG engine**: the retriever imposes a "source → chunks" model (one entity = one `books` row) — real but bounded genericity.

13. Technical stack (inventory)

Languages: Python 3.11 · TypeScript/React · Swift · C++17/JUCE (audio). **Inference**: MLX / `mlx_lm` 0.31.3 · `mlx_vlm` · `mlx-audio` · `sentence-transformers` · Ollama (legacy embeddings/healthcheck). **Models**: Qwen3.6-35B-A3B (brain) · Qwen3-Coder-30B-A3B (coder) · Qwen2.5-VL-7B (vision) · bge-m3 (embeddings)

· Qwen3-TTS / Irodori / MOSS (TTS) · whisper-large-v3-turbo (ASR, cached). **Web / services**: FastAPI · Uvicorn · Starlette · httpx · WebSocket · Tauri v2 · Tailwind v4. **Data**: SQLite + **FTS5** + **sqlite-vec** · Pydantic v2. **Audio**: ACE-Step · RVC · librosa · soundfile · pyloudnorm. **Tools/agent**: **MCP** protocol · hand-rolled ReAct loop. **Ops**: `launchd` (RunAtLoad/KeepAlive) · dedicated venvs · pytest · Git (Conventional Commits, protected branches).

---

14. Glossary

- **Gateway** — the single inference endpoint (`:8090`) that owns the workers and routes by model.
- **Worker** — an `mlx_lm.server` / `mlx_vlm.server` process serving **one** model.
- `pinned` — a permanently resident model, never evicted (the `brain`).
- **LRU eviction** — freeing RAM by removing the least-recently-used worker.
- `inflight` — number of in-progress requests on a worker (never evicted if > 0).
- **Adoption** — detecting and taking over an already-launched worker at gateway startup.
- `reserve` — freeing RAM ahead of time for an external audio job.
- `klody_memory` — the reusable RAG package extracted from Library Brain.
- **ReAct** — the *Reason* + *Act* agent loop (reason, then call a tool).
- **MCP** — *Model Context Protocol*, the standard for exposing tools to agents.
- **Gate / eval** — automatic verification that the **models** still answer well (≠ the code's unit tests).

Portfolio document — a synthesis of the Klody architecture, from its genesis (May 9, 2026) to its state on June 13, 2026.