



BUILDING A PRIVATE AI RESEARCH HOMELAB

and how to document it so an LLM can help you build the next one

WHAT YOU'RE LOOKING AT

Eight-node cluster. Three inference tiers. Zero cloud dependency.

8

nodes

~\$2K

hardware total

18

tok/s on Tier 3

Pi 5 cluster + Hailo-10H NPUs + Minisforum N5A NAS + Jetson Orin Nano + Pi Zero W ticker. All running in a 3D-printed rack in a closet.



Why this exists

Not a hobby project. Real motivations.

Cloud LLM costs scale with experimentation

Running 1000 prompts to characterize a model's behavior on edge hardware costs real money on a hosted API. Local inference makes systematic experimentation cheap.

Hosted APIs hide the substrate

You can't measure prompt eval vs. token gen latency on a specific quantization on specific silicon if your only interface is a chat completions endpoint.

Edge AI is real research

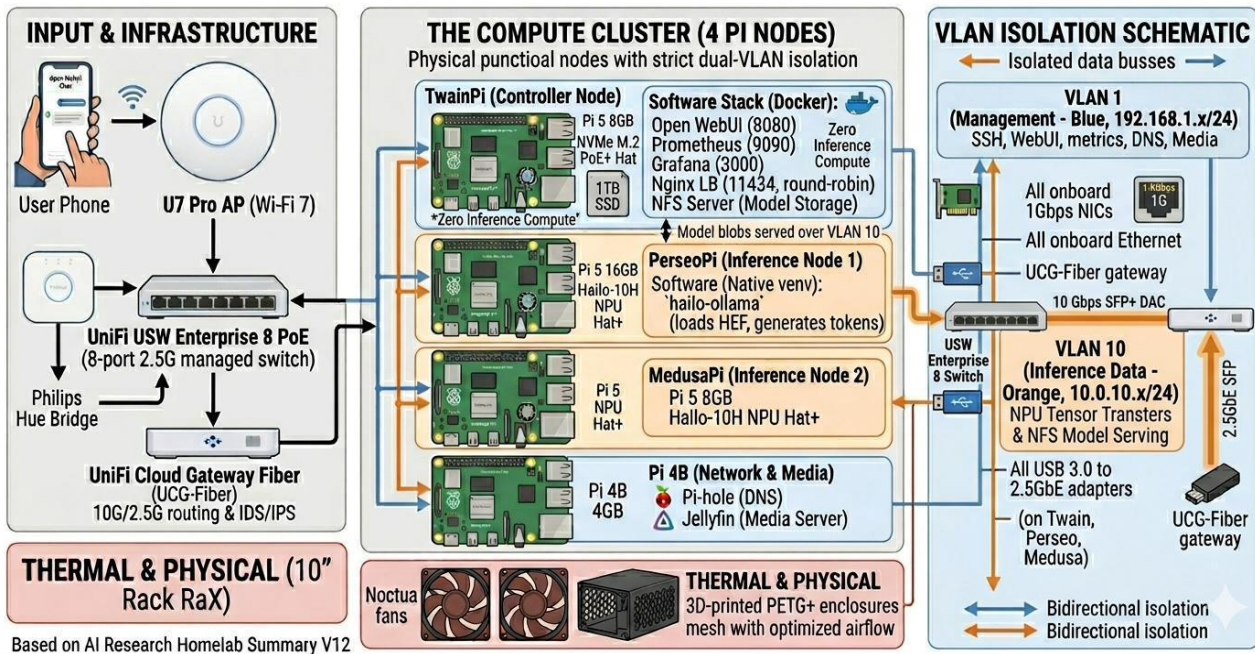
Real economic implications. Requires hands-on access to actual hardware constraints — memory envelopes, thermal limits, quantization trade-offs.

Privacy

Briefly. The audience knows. Nothing leaves the closet unless I send it.

The architecture in one diagram

PRIVATE AI HOMELAB: FULL STACK ARCHITECTURE V12



Three inference tiers · one frontend · one observability stack · two physical visualization surfaces.

Three inference tiers — heterogeneity is the value

Each tier plays to its hardware's strengths. The cluster's value is in the asymmetry.

Tier 1 — NPU

PerseoPi + MedusaPi (Pi 5 + Hailo-10H)

HEF-compiled sub-2B models

On-chip SRAM, sub-2W per node

Best for: classification, embeddings, fast small-model generation

Tier 2 — GPU

Narnia (Minisforum N5A, AMD 780M, ROCm)

Gemma 4 E4B at effective Q8 (~9.6 GB)

~15 GB GTT, 18.4 tok/s output

Best for: quality-first reasoning, longer responses, vision

Tier 3 — CUDA

Elroy (Jetson Orin Nano 8GB)

qwen2.5:3b · gemma3:4b-it-qat ·
nemotron-3-nano:4b

~5 GiB effective model budget, ~18 tok/s

Best for: speed-first chat, fast iteration,
Mamba-Transformer hybrid

What this enables

Four research directions the substrate uniquely makes possible.

Multi-NPU Pipeline Parallelism

Partition large models across PerseoPi + MedusaPi via zero-copy VLAN 10 sockets. Expert-wise MoE partitioning is the most likely first target.

Predictive Thermal Orchestration


Replace reactive thermal-check.sh with a DRL agent on Prometheus metrics, including ArgusPi ambient telemetry.

Speculative Decoding (asymmetric)

Drafter on one Hailo NPU + verifier on Narnia E4B (or Elroy if Phase 5.4 lands first). Ready to attempt — substrate is in place.

Mixture-of-Agents semantic routing

Classifier design + integration with Nginx LB. Hard prereq: all four inference tiers stable and monitored.



None of this is possible to build and maintain solo without a working partner.

The traditional answer is a team. The answer that's emerged in the last year is a different kind of partner — one that operates on documentation.

That's what the rest of the talk is about.

Setting more BIOS VRAM made GPU memory smaller

On AMD APUs, ROCm's GTT (Graphics Translation Table) is capped at 50% of OS-visible RAM. Reserving more in BIOS reduces what the OS can see — which halves the GTT cap.

Counterintuitive answer:

Set BIOS UMA to AUTO. Let the kernel see all 32 GB.

Effective GPU memory grows from 8 GB → 15 GB.

```
# Before — BIOS UMA = 16 GB locked
$ rocm-smi --showmeminfo gtt
GTT Total Memory: 8.0 GB
GTT Used Memory: 6.2 GB

# Gemma 4 E4B fits at ~6 tok/s,
# OOMs intermittently under load.

# After — BIOS UMA = AUTO
$ rocm-smi --showmeminfo gtt
GTT Total Memory: 15.2 GB
GTT Used Memory: 9.6 GB # full E4B resident

# Same model, same prompts — 18.4 tok/s
# 3x throughput, no OOM.
```


Latent for a month. Diagnosed in one command.

The symptom

jetson-exporter on :9835 was reported as 'enabled' by systemd, but every reboot left it Active: inactive (dead). Manual systemctl start worked fine. Unit-level journal was empty.

The catch

Discovered only because HeraldPi rendered "--" for the GPU panel for hours. Software dashboards alone hadn't caught it — Prometheus had no up==0 alert rule.

Diagnostic fingerprint of a deliberately-skipped unit:

```
$ systemctl show jetson-exporter.service | grep -E 'Result|Restarts|Active|Condition|Assert'
Result=success           # not 'failure' – the smoking gun
NRestarts=0              # never started
ActiveEnterTimestamp=n/a # never went active
ConditionResult=no       # systemd deliberately skipped this unit
AssertResult=no
```

The command that found it

Cycle messages log at the target level, not the unit level. `journalctl -u <unit>` shows nothing. `journalctl -b` shows everything.

```
$ sudo journalctl -b | grep -i "ordering cycle"

Mar 24 06:52:29 Elroy systemd[1]: multi-user.target:
    Found ordering cycle on jetson-exporter.service/start
Mar 24 06:52:29 Elroy systemd[1]: multi-user.target:
    Found dependency on jtop.service/start
Mar 24 06:52:29 Elroy systemd[1]: multi-user.target:
    Job jetson-exporter.service/start deleted to break ordering cycle
    starting with multi-user.target/start
```

Root cause: jetson-stats v4.3.2 ships jtop.service with both `After=multi-user.target` AND `WantedBy=multi-user.target`. Any service that adds `After=jtop.service` closes the cycle.



PHYSICAL OBSERVABILITY

Software dashboards alone wouldn't have caught it.

HeraldPi's 16×32 LED ticker rendered "--" for the GPU panel for hours. That visible degradation prompted manual investigation. A Grafana dashboard nobody opens cannot raise an alarm.

The principle:

*Everything worth measuring is worth surfacing
somewhere visible.*



That principle applies to documentation, too.

Everything worth doing is worth measuring.

Everything worth measuring is worth surfacing visibly.

Everything worth knowing about your project is worth writing down.

THE PREMISE

The LLM is not the substrate.

The documentation is the substrate.

The LLM operates on it.

You curate it. You version it. You maintain its accuracy. The LLM's job is to extend, query, and use it — not to be the source of truth.

The three-layer model

This is the conceptual spine. If you remember nothing else, remember the layers.

Layer 1 — Operational documentation

Versioned PDFs. Per-deployment-phase guides. Per-node user guides. A central roadmap. A drift log. You own the final content.

Layer 2 — System prompt

The LLM's runtime configuration. Tells the LLM how to interpret your library: which versions are authoritative, which are historical, what role to play.

Layer 3 — The working chat

Where the back-and-forth happens. Ephemeral by design. Output is (a) a substrate change, (b) a doc update, or (c) both. Anything else didn't accomplish anything.

Layer 1 — Operational documentation

14 versioned PDFs in a month. Long, dense, and exactly as detailed as they need to be.

Project Roadmap V3.8

Strategic overview, phase summaries, drift log

User Guide V9.1

Day-to-day operator reference, SOPs, troubleshooting

Phase 2 / 3 / 5 Deployment Guides

How each tier was built and why

Elroy / ArgusPi / HeraldPi User Guides

Per-node operations and learnings

Pre-Research Priorities V1.1

Backlog: every open item, ordered

Written by you, with the LLM's help. You own the final content.

Layer 2 — The system prompt

How the LLM reads your library. The hierarchy that makes EPOCH-based versioning work.

```
# Excerpted from the project's system prompt:
```

1. EPOCH 2 documents are AUTHORITATIVE.
Any document with 'EPOCH 2' or version \geq V3.6 supersedes prior content.
2. EPOCH 1 documents are HISTORICAL CONTEXT ONLY.
They describe how things were built. Not necessarily current reality.
3. The DRIFT LOG is the reconciliation layer.
Lists every known inconsistency between EPOCH 1 docs and current state.
4. When EPOCH 1 and EPOCH 2 conflict on operational facts, EPOCH 2 wins.

Without this, the LLM doesn't know whether V3.5 or V3.8 of the roadmap is the live document. With it, conflicts resolve automatically.

Layer 3 — The working chat

Ephemeral by design.

The output of every chat is one of three things:



A change to the deployed substrate

(new config, new node, new model, new code shipped)



An update to a Layer 1 document

(operational learning captured before context fades)



Both

(the most common — substrate change AND its documentation)

If a chat produces neither (a), nor (b), nor (c), it didn't accomplish anything durable.

Practical mechanics — 1 of 3

Things you can implement Monday morning.

1

Use a tool with persistent project context

Upload your operational docs into a project knowledge base. The LLM searches them on every relevant query. (The methodology is what matters, not the vendor.)

2

Replace, don't accumulate

When V1.1 supersedes V1.0, V1.0 leaves the project. Stale versions in the search corpus produce stale answers. Exception: keep one snapshot per epoch boundary as historical context.

3

Surgical revisions over rebuilds

Most updates should be additive — V2.0 → V2.1 with a change log, preserved structure, surgical inserts. Rebuild only when so much changed that incrementing in place would mislead readers.

Practical mechanics — 2 of 3

The two habits that matter most. Skip these and the methodology unravels.

4. End every session by capturing what just happened

If a session produced operational learnings, they go into the relevant document BEFORE the chat ends. If you're tired, capture a checklist for the next session — but never let findings escape into chat-only state. Pre-Research Priorities §0a "Completed Since V1.0" was authored at the end of the same session that produced its 8 findings.

5. Maintain a drift log

A single section in your central document that catalogs every known inconsistency between what's documented in older versions and what's true now. The LLM uses this as a reconciliation layer when older docs conflict with newer ones. Roadmap V3.8 §10 is the example.

Practical mechanics — 3 of 3

Honesty about the partnership. The system prompt itself is a living document.

6. Be honest about what the LLM is and isn't doing

It's not making architectural decisions. It's not picking which research questions matter. It IS helping you write code, draft documentation, debug systemd units, and being a tireless documentation-aware reviewer. Calibrate your framing accordingly.

7. Treat the system prompt as a living document

It's part of the substrate. New epoch boundaries, new conventions, new authoritative versions land in the prompt — not just in the documents. The system prompt tells the LLM HOW to read your library. Revisit it roughly every major doc cascade.

What goes wrong if you skip these steps

Four specific failure modes. Concrete, observable, preventable.

Documentation drift

You change something in production but never update docs. Six months later, the LLM is reasoning about a system state that no longer exists.

Context window starvation

Every new chat starts from zero. You spend 30 minutes re-explaining your setup and the LLM still misses critical details.

Stale-version contamination

Old versions of docs in the search corpus produce confidently-wrong answers.

Hallucinated history

Without a drift log, the LLM makes up plausible-sounding reasons for past decisions when asked.

Step 1 — Input state

April 27, 2026, ~10am. What the LLM saw at session start:

Project knowledge base:

- Roadmap V3.8 (EPOCH 2 authoritative)
- User Guide V9.1
- Pre-Research Priorities V1.0
- Elroy User Guide V2.0
- Phase 5 Deployment Guide V1.1
- + EPOCH 2 doctrine in the system prompt

User message:

"Nemotron just came out, can it run on Elroy?"

Working chat — what actually happened

```
$ ssh USER@elroy
$ ollama pull nemotron-3-nano:4b          # ~2.8 GB Q4_K_M

$ curl -s http://localhost:11434/api/generate \
  -d '{"model":"nemotron-3-nano:4b","prompt":"Who are you?"}'
> "I'm Qwen, a large language model developed by Alibaba Cloud..."
# ↑ identity bleed: distilled from Qwen reasoning traces

$ cat > nemotron-elroy.modelfile <<'EOF'
FROM nemotron-3-nano:4b
PARAMETER num_ctx 2048
SYSTEM ""You are Nemotron 3 Nano, an open-weights small language model
created by NVIDIA, running locally on a Jetson Orin Nano edge device.
Do not claim to be Qwen, ChatGPT, or any other model.""
EOF
$ ollama create nemotron-3-nano:4b-elroy -f nemotron-elroy.modelfile

# ...and then HeraldPi's GPU panel rendered "--" — exporter was dead.
```

Two findings in one session: a new model deployed with identity-anchoring SYSTEM prompt, and a previously-unknown month-old systemd outage surfaced and fixed.

Capture moment — at end of session

Two documents updated. Eight findings recorded. Not later — now.

Pre-Research Priorities V1.0 → V1.1

§0a "Completed Since V1.0" — 8 rows:

- Nemotron Q4_K_M deployed
- Nemotron Q8_0 deployed (swap-bound)
- Identity-bleed mitigation pattern
- -elroy alias convention formalized
- jetson-exporter unit hardened
- jetson-stats v4.3.2 cycle diagnosed
- Silent-skip diagnostic technique
- Jetson zram swap recovery

Elroy User Guide V2.0 → V2.1

Surgical additions:

- §2.1 model table — Nemotron rows
- §3.3 Mamba KV cache caveat
- §10.3 hardened jetson-exporter unit
- §11.3 -elroy alias convention
- §11.4 identity bleed and SYSTEM prompts
- §13.7 silent-skip diagnostic
- §13.8 zram swap recovery

By the next day, three of those findings would already be partially forgotten.

Output state — project knowledge after the session

End of session: stale docs out, new docs in. Single replacement. Now everyone — me, the LLM, future-me next month — sees the new ground truth.

REMOVED

- Pre-Research Priorities V1.0
- Elroy User Guide V2.0

(Intra-EPOCH revisions. No reason to keep stale versions in the search corpus.)

ADDED

- Pre-Research Priorities V1.1
- Elroy User Guide V2.1

Next chat opens with full context — Nemotron deployment, alias convention, ordering-cycle fix. No manual passing.

This is what "persistent context window the LLM can read" looks like in practice.

Anti-patterns I've fallen into

Honest about what doesn't work. The methodology took time to develop and includes real failure modes.

DON'T

"Let the LLM write the docs from scratch"

It produces plausible-looking documentation that doesn't match reality. Documents must originate from your knowledge of what's actually deployed; the LLM helps you structure, expand, and maintain them. Inversion of authorship is the #1 failure mode for people new to this workflow.

DON'T

"Skip the system prompt; just paste context every time"

Works for two weeks. Falls apart by month two. Without a system prompt establishing how to interpret the documentation library, you end up re-explaining the project's structure every session — which negates the value of having documentation.

DON'T

"Capture findings later when I have time"

Findings always look smaller in retrospect than they did in the moment. The discipline is NOW, while context is fresh. By tomorrow, three of today's eight findings are partially forgotten.

The methodology generalizes

This talk has been homelab-specific. The methodology isn't.

Software projects

Architecture decision records, API docs, runbooks. The drift log is the changelog.

Research notebooks

Hypotheses, methods, findings. Each finding has a date, a context, and a citation.

Legal cases

Filings, exhibits, correspondence. Versioned by motion. Drift log captures every factual revision.

Any multi-month effort

Where context accumulates faster than human memory holds it. The homelab is just the most concrete example.

If it works for a research-grade AI cluster maintained by one person, it works for your own multi-month project.



THE TAKEAWAY

"The LLM didn't build my homelab. I built it.

The LLM helped me document it well enough that I could keep building it through the month without losing track of why I made the decisions I made."

That's the actual capability. Not generation, but persistence.

Questions?