Summary

OpenClaw Memory Enhancement Stack

Four tools that address different layers of OpenClaw's memory problem. Each solves a different failure mode — they're complementary, not competing.

Tool	What It Does	Runs On	Status
QMD	Hybrid search (BM25 + vector + LLM rerank) over markdown files	Linux agent box (CPU) or Mac (Metal)	Stable — v1.1.6
memory-lancedb-pro	Vector memory plugin with decay, hybrid retrieval, scope isolation	Linux agent box, embeddings on Mac	Stable — npm package
OpenViking	Context database with virtual filesystem and tiered token loading	Linux agent box, VLM calls on Mac	⚠ Hold — LiteLLM supply chain attack
MetaClaw	Continual learning proxy — agent gets smarter over time	Linux agent box, forwards inference to Mac	Beta — v0.4.0, very new

---

QMD — Search That Actually Works

Created by: Tobi Lütke (Shopify CEO)

GitHub: github.com/tobi/qmd
License: MIT
Install: `npm install -g @tobilu/qmd`

What it solves: OpenClaw's built-in `memory_search` uses pure vector embeddings that routinely return wrong results. QMD replaces it with a three-stage hybrid pipeline.

How it works:

1. A local 1.7B LLM expands your query into two alternative formulations

2. All three variants run through BM25 keyword search AND vector cosine similarity in parallel
3. Results merge via Reciprocal Rank Fusion (original query weighted 2×)
4. Top 30 candidates scored by a local reranker model
5. Final ranking blends RRF and reranker scores with position-aware weighting

Local models (~2.5GB total, auto-downloaded):

- embeddinggemma-300M (embedding)
- Qwen3-Reranker-0.6B (reranking)
- qmd-query-expansion-1.7B (query expansion — Tobi's custom fine-tune)

Remote inference: Not supported natively. Models run via node-llama-cpp in-process. On 128GB Mac Studio they're negligible. On the Linux agent box they run fine on CPU (~10-15s per query vs ~3-5s on Metal).

OpenClaw integration: Set `memory.backend = "qmd"` in openclaw.json. OpenClaw creates collections and indexes automatically on boot.

Key commands:

```
qmd query "what did I decide about the camera setup"   # hybrid search (best quality)
qmd search "frigate RTSP"                              # keyword only (fastest)
qmd vsearch "home automation preferences"              # vector only
qmd status                                             # index info
qmd update --pull                                      # re-index (git pull first)
qmd embed -f                                           # force re-embed all
```

---

memory-lancedb-pro — Long-Term Memory With Decay

Created by: CortexReach (community)

GitHub: github.com/CortexReach/memory-lancedb-pro
License: MIT
Install: `npm i memory-lancedb-pro`

What it solves: OpenClaw's default memory has no decay (everything stays equally weighted forever), no hybrid search (vector only), no deduplication, and MEMORY.md dumps its entire contents into every session wasting tokens.

How it works:

- **Auto-capture:** Extracts preferences, facts, decisions, and entities from conversations automatically (up to 3 per turn)
- **Auto-recall:** Injects relevant memories into context before agent responds (up to 3 entries)
- **Smart extraction:** LLM-powered 6-category classification (Profile, Preferences, Entities, Events, Cases, Patterns) with L0/L1/L2 metadata
- **Hybrid retrieval:** Vector search + BM25 keyword search fused with RRF, then cross-encoder reranking
- **Weibull time decay:** Memories that aren't accessed gradually fade from active retrieval
- **Three-tier system:** Core → Working → Peripheral with automatic promotion/demotion
- **Multi-scope isolation:** global, agent:<id>, project:<id>, user:<id>, custom:<name>

Remote inference: Yes — embedding uses any OpenAI-compatible `/v1/embeddings` endpoint. Point `baseUrl` at Ollama/LM Studio on the Mac. Reranking uses Jina, SiliconFlow, or any compatible reranker API.

Key config (openclaw.json):

```
{
  "plugins": {
    "slots": { "memory": "memory-lancedb-pro" },
    "entries": {
      "memory-lancedb-pro": {
        "enabled": true,
        "config": {
          "embedding": {
            "model": "nomic-embed-text",
            "baseUrl": "http://<mac-headscale-ip>:11434/v1",
            "apiKey": "not-needed"
          },
          "autoCapture": true,
          "autoRecall": true,
          "smartExtraction": { "enabled": true },
          "retrieval": { "mode": "hybrid", "vectorWeight": 0.7, "bm25Weight": 0.3 }
        }
      }
    }
  }
}
```

Key commands:

```
openclaw memory-pro list --scope global --limit 20      # list memories
openclaw memory-pro search "query" --scope global       # search memories
openclaw memory-pro stats --scope global                # count, categories, age
openclaw memory-pro export --output memories.json       # backup
openclaw memory-pro import memories.json --dry-run      # import (test first)
openclaw memory-pro reembed                             # after changing embedding model
openclaw memory-pro delete <id>                         # delete specific memory
openclaw memory-pro delete-bulk --before "2026-01-01"   # bulk delete
openclaw memory-pro migrate check --source /path/to/old  # migrate from built-in plugin
```

After config changes: `openclaw config validate` then `openclaw gateway restart`.
After editing plugin .ts files: `rm -rf /tmp/jiti/` then restart (jiti caches stale compiled code).

OpenViking — Context Database With Virtual Filesystem

Created by: ByteDance / Volcengine Viking Team

GitHub: github.com/volcengine/OpenViking
License: Apache 2.0
Stars: ~17,900

⚠ **DO NOT INSTALL RIGHT NOW.** OpenViking has a dependency on `litellm>=1.0.0`. LiteLLM was hit by a supply chain attack on March 24, 2026 (TeamPCP backdoored versions 1.82.7-1.82.8). The entire litellm package is currently quarantined on PyPI — fresh installs of anything depending on it will fail. Wait for the quarantine to lift, then install with `"provider": "openai"` pointing at your local Ollama endpoint to bypass LiteLLM entirely.

What it solves: Replaces flat vector storage with a hierarchical virtual filesystem. Instead of dumping everything into embeddings and hoping retrieval works, OpenViking organises context into structured directories accessible via `viking://` URIs.

How it works:

- **Virtual filesystem:** Three root directories — `viking://resources/` (documents, repos), `viking://user/` (preferences, habits), `viking://agent/` (skills, task memories)
- **Unix-like navigation:** `ls`, `find`, `read`, `tree`, `grep` against agent context
- **L0/L1/L2 tiered loading:** L0 = ~100 tokens (abstract), L1 = ~2,000 tokens (overview), L2 = full content on demand. Claims 91-95% token cost reduction vs traditional RAG
- **Automatic memory self-iteration:** At session end, extracts 6 memory categories and updates the appropriate directories

Remote inference: Yes — configure `"provider": "openai"` with `"api_base"` pointing at the Mac's Ollama. This bypasses the LiteLLM dependency entirely for your use case.

Benchmark results with OpenClaw:

- Task completion: 52.08% (vs 35.65% native OpenClaw — 43% improvement)
- Input tokens: 4.3M (vs 24.6M native — 91% reduction)

When to install: After LiteLLM quarantine lifts. Clone repo, strip `litellm>=1.0.0` from pyproject.toml if needed, use `openai` provider only.

MetaClaw — The Agent That Learns From Its Mistakes

Created by: AIMING Lab, UNC Chapel Hill

GitHub: github.com/aiming-lab/MetaClaw
License: Apache 2.0

Paper: arXiv 2603.17187 (ranked #1 on HuggingFace Daily Papers, March 18 2026)
Stars: ~2,700

What it solves: The other three tools store and retrieve memories. MetaClaw is the only tool that actually makes the agent *smarter over time*. It learns from failure patterns and generates corrective behavioural skills.

How it works: MetaClaw sits as an OpenAI-compatible proxy between OpenClaw and your LLM. It intercepts every interaction.

- **Skill-driven fast adaptation (gradient-free):** Analyses failure trajectories via an LLM evolver and synthesises new behavioural skills that take effect immediately. No GPU needed, no downtime. If your agent repeatedly makes the same mistake, MetaClaw generates a corrective skill.
- **Opportunistic policy optimisation (gradient-based, optional):** Cloud LoRA fine-tuning via RL, triggered only during user-inactive windows. An Opportunistic Meta-Learning Scheduler monitors sleep hours, keyboard inactivity, and calendar to find training windows. Can be skipped entirely.

Results: Skill-driven adaptation improved accuracy by up to 32% relative. Full pipeline advanced Kimi-K2.5 from 21.4% to 40.6% accuracy (approaching GPT-5.2's 41.1%).

Remote inference: Yes — MetaClaw IS a proxy. Run it on the Linux box, point its upstream at the Mac's LM Studio/Ollama endpoint. All inference happens on the Mac. MetaClaw just intercepts and learns.

For your setup: Use skills-only mode (gradient-free). No GPU needed on the Linux box. Since v0.3.3, MetaClaw has a native OpenClaw plugin. Since v0.4.0 (March 25), it includes a "Contexture layer" for cross-session memory.

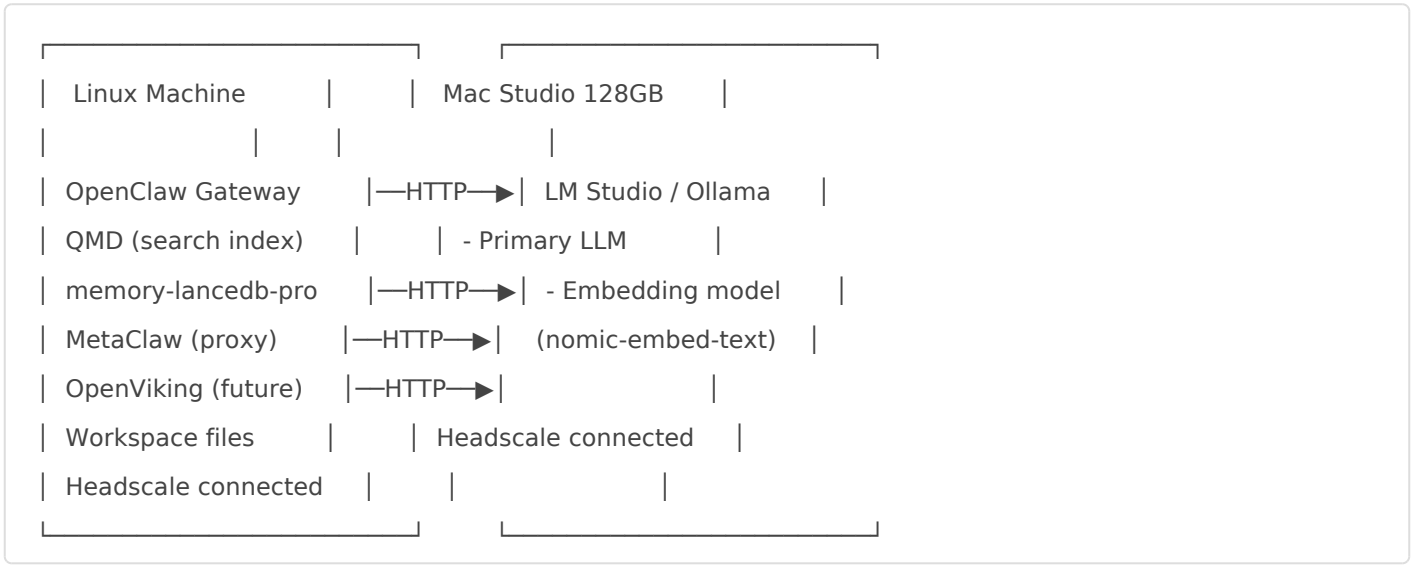
Caveat: Very new (16 days old, 7 releases). Academically rigorous but least battle-tested of the four tools. Worth running but expect rough edges.

Install Order

1. **QMD** — `npm install -g @tobilu/qmd`, set `memory.backend = "qmd"`. Immediate improvement to search/recall.
2. **memory-lancedb-pro** — `npm i memory-lancedb-pro`, configure embedding endpoint to Mac. Fixes memory bloat and adds decay.
3. **MetaClaw** — Install as OpenClaw plugin, configure as proxy. Adds learning over time.
4. **OpenViking** — Wait for LiteLLM situation to resolve. Then install with `openai` provider pointing at Mac.

Test each layer before adding the next. If something breaks, you know which component caused it.

Architecture



All heavy inference on the Mac. All files, indexing, gateway, and agent logic on the Linux box.

Revision #1
Created 26 March 2026 09:36:35 by Conor
Updated 26 March 2026 09:36:45 by Conor