QMD

QMD — Local Hybrid Search for OpenClaw Agent Memory

What it is: QMD is a fully local, on-device search engine for markdown files built by Tobi Lütke (Shopify CEO). It replaces OpenClaw's broken built-in memory search with a three-stage hybrid pipeline: BM25 keyword matching + vector semantic search + LLM re-ranking. No API keys, no cloud calls, no network traffic after initial model download.

Why it matters: OpenClaw's default `memory_search` uses pure vector embeddings that routinely return semantically adjacent but factually wrong results. QMD fixes this by running three search methods in parallel and fusing them with Reciprocal Rank Fusion, then re-ranking the top candidates with a local LLM. Community reports claim it cuts token usage by 95%+ compared to context-stuffing approaches.

Install: `npm install -g @tobilu/qmd` (requires Node.js 22+ or Bun). Three GGUF models (~2.5GB total) auto-download from HuggingFace on first use.

How It Works

Every query goes through this pipeline:

1. **Query Expansion** — A local 1.7B LLM generates two alternative formulations of your query
2. **Parallel Search** — All three query variants (original weighted 2×) run through both BM25 full-text search AND vector cosine similarity search simultaneously
3. **Reciprocal Rank Fusion (RRF)** — Results merge with k=60, original query weighted 2×, top-rank bonus (+0.05 for #1, +0.02 for #2-3)
4. **LLM Re-ranking** — Top 30 candidates scored by a local reranker model (yes/no + logprob confidence)
5. **Position-Aware Blending** — Final ranking blends RRF and reranker scores: ranks 1-3 use 75% RRF / 25% reranker, ranks 4-10 use 60/40, ranks 11+ use 40/60

Documents are chunked into ~900-token pieces with 15% overlap, using smart boundary detection that prefers markdown headings. Embeddings and LLM responses are cached in SQLite (`~/.cache/qmd/index.sqlite`) with content-hash keying, so moving/renaming files doesn't require re-embedding.

Local Models

QMD runs three GGUF models locally via `node-llama-cpp`. All models auto-download to `~/.cache/qmd/models/` on first use:

Purpose	Model	Size	Source
Embedding	embeddinggemma-300M (Google)	~300MB	HuggingFace
Re-ranking	Qwen3-Reranker-0.6B-Q8_0	~600MB	HuggingFace
Query Expansion	qmd-query-expansion-1.7B (Tobi's fine-tune)	~1.7GB	HuggingFace

On Apple Silicon, QMD auto-detects Metal GPU acceleration at startup. Total VRAM/memory footprint is ~2.5GB — negligible on a 128GB Mac Studio.

Swapping the Embedding Model

The embedding model can be overridden via environment variable:

```
# Use Qwen3-Embedding for better multilingual (CJK) support
export QMD_EMBED_MODEL="hf:Qwen/Qwen3-Embedding-0.6B-GGUF/Qwen3-Embedding-0.6B-Q8_0.gguf"

# After changing model, re-embed all collections:
qmd embed -f
```

Note: This still loads a local GGUF file — it doesn't point at a remote API. Vectors are not cross-compatible between models, so you must re-index after switching.

Can I Point Models to a Remote Machine?

Short answer: No — QMD is designed to be fully local.

QMD runs all inference through `node-llama-cpp` in-process. There is no built-in configuration to route embedding, reranking, or query expansion to a remote API endpoint. The project's tagline is literally "Tracking current sota approaches while being all local."

There is a `QMD_OPENAI_BASE_URL` environment variable referenced in some third-party integration guides (specifically the `ehc-io/qmd` fork), but this is **not part of Tobi's original QMD** and applies to a different use case.

Workarounds if you need remote inference:

- The models are tiny (~2.5GB total). On a 128GB Mac Studio they're negligible — just run them locally alongside your main LLM
- If you absolutely need to offload: QMD exposes an MCP server (`qmd mcp`) with HTTP transport mode. You could run QMD on a remote machine and connect to it as an MCP service. But the models themselves still run local to wherever QMD is installed
- Fork and modify — QMD is MIT licensed and the embedding/reranking code is in `src/`. You could swap the `node-llama-cpp` calls for HTTP calls to a remote endpoint, but this is custom development

---

OpenClaw Integration

Three integration paths, from simplest to most flexible:

1. Built-in Memory Backend (Recommended)

Set `memory.backend = "qmd"` in `~/.openclaw/openclaw.json`:

```
{
 "memory": {
  "backend": "qmd",
  "citations": "auto",
  "qmd": {
   "includeDefaultMemory": true,
   "command": "qmd",
   "searchMode": "search",
   "update": {
    "interval": "5m",
    "debounceMs": 15000,
    "onBoot": true,
    "waitForBootSync": false
   },
   "limits": {
    "maxResults": 6,
    "timeoutMs": 4000
   },
```

```
      "scope": {
        "default": "deny",
        "rules": [
          { "action": "allow", "match": { "chatType": "direct" } }
        ]
      }
    }
  }
}
```

OpenClaw automatically creates collections (e.g. `memory-root` for `memory/**/*.md`) and indexes on boot.

2. MCP Server

Run `qmd mcp` to expose QMD as an MCP tool. Supports stdio and HTTP transport. HTTP daemon mode keeps models warm in VRAM between queries, reducing latency from ~16s to ~10s.

3. openclaw-engram Plugin

A community plugin (github.com/joshuaswarren/openclaw-engram) that uses QMD as the backend for a persistent long-term memory system with LLM-powered extraction.

Search Modes

Command	Method	Speed	Best For
`qmd search`	BM25 keyword only	~50-200ms	You know the exact terms
`qmd vsearch`	Vector similarity only	~500-1000ms	Semantic matches without reranking
`qmd query`	Full hybrid pipeline	~3-10s	Highest quality results (recommended)

Key Features

- **Query documents** — Structured multi-line queries with typed lines (`lex:`, `vec:`, `hyde:`) combining keyword precision with semantic recall
- **Intent parameter** — Optional `--intent` flag disambiguates queries across the entire pipeline. "performance" means different things in different contexts
- **Quoted phrases and negation** — `"C++ performance" -sports -athlete` works in lex search

- **Content-hash keying** — Moving/renaming files doesn't require re-embedding
- **LLM response caching** — Query expansion and rerank scores cached in SQLite
- **Agentic output formats** — `--json`, `--files`, `--md`, `--full` for structured agent consumption
- **Collection contexts** — Hierarchical semantic metadata (e.g. "Personal notes and meeting logs") improves relevance
- **Separate indexes** — `qmd --index work search "quarterly reports"` keeps knowledge bases isolated

Storage

- **Index:** `~/.cache/qmd/index.sqlite` (SQLite with FTS5 + sqlite-vec)
- **Models:** `~/.cache/qmd/models/` (~2.5GB)
- **Config:** `~/.config/qmd/index.yml` (collection definitions, respects XDG_CONFIG_HOME)

Quick Reference

```
# Install
npm install -g @tobilu/qmd

# Create a collection
qmd collection add ~/.openclaw/workspace --name workspace --mask "**/*.md"

# Generate embeddings
qmd embed

# Search
qmd query "what did I decide about the camera setup"

# Status
qmd status

# Re-index (with git pull for remote repos)
qmd update --pull
```

Version: v1.1.6 (current as of March 2026)
License: MIT
GitHub: github.com/tobi/qmd
npm: `@tobilu/qmd`
Runtime: Node.js 22+ or Bun

Dependencies: node-llama-cpp, sqlite-vec, better-sqlite3

Revision #1
Created 26 March 2026 09:10:10 by Conor
Updated 26 March 2026 09:14:11 by Conor