Building Fault-Tolerant Memory for OpenClaw AI Agents

A Three-Layer Architecture for the SQLITE_CANTOPEN Regression in 2026.3.x

A Production Deployment Case Study

Adnan Tanveer (Addy)
Speedrun AI Labs
Sydney, Australia
speedrunlab.ai

March 16, 2026 | Paper Version 1.1, submitted to arXiv cs.SE

NOTE ON TIME-SENSITIVITY: This paper documents the state of OpenClaw as of March 12, 2026 (version 2026.3.2, build 85377a2). OpenClaw is an actively developed project with near-daily releases. The specific bugs, configurations, and workarounds described here may change in future versions. The architectural principles (redundant memory layers, graceful degradation, independence from single-point-of-failure components) remain applicable regardless of version. Readers should verify version compatibility before applying these configurations.

Abstract

OpenClaw, the open-source autonomous AI agent framework with over 247,000 GitHub stars, ships with a memory system that exhibits recurring reliability failures. Between January and March 2026, the project's issue tracker documented persistent SQLite database errors, dirty memory indexes after updates, complete persistence failures during active sessions, and a critical jiti module resolver regression in version 2026.3.1 that broke native SQLite bindings entirely. These failures cause agent crash loops, lost context during long coding sessions, and silent memory corruption during compaction cycles.

This paper presents a three-layer fault-tolerant memory architecture that eliminated these failures in our deployment and architecture review while preserving full memory persistence across session boundaries. We make three contributions: (1) we catalog five distinct failure modes in OpenClaw's memory system from public issue trackers; (2) we design and implement a three-layer architecture combining pre-compaction memory flushing, QMD-based local hybrid search, and Mem0 self-hosted on Qdrant; and (3) we validate the architecture in a production deployment on a single Mac Mini. This is a practitioner experience report from a production deployment.

We document the complete diagnostic and deployment process, provide reproducible configuration templates with sanitised credentials, and verify persistent memory across session resets on OpenClaw 2026.3.2. The architecture runs on a single Mac Mini (M4, 24GB RAM) with zero cloud dependency for search and minimal API cost for embeddings.

Keywords: OpenClaw, AI agents, fault-tolerant memory, Mem0, QMD, Qdrant, SQLite regression, autonomous agents, memory persistence

1. Introduction

OpenClaw (formerly Clawdbot, Moltbot) is an open-source autonomous AI agent framework created by Peter Steinberger in November 2025. It enables locally-hosted AI agents to execute tasks through messaging platforms including Telegram, WhatsApp, Slack, Discord, and iMessage. The project achieved viral adoption in January 2026, accumulating over 247,000 GitHub stars and 55,000 forks by March 2026 [1].

On February 14, 2026, Steinberger announced he was joining OpenAI, with OpenClaw transitioning to a foundation structure with OpenAI sponsorship [2]. Development continued at a near-daily release cadence with growing contributor counts (93 or more in version 2026.3.2, rising to 194 or more by version 2026.3.7) [3].

Despite this active development, OpenClaw's memory system exhibited recurring reliability problems throughout its rapid growth period. The default memory architecture relies on plain Markdown files as the source of truth, with a SQLite-backed vector index for semantic search. This architecture proved fragile under production workloads, particularly for long-running coding sessions where context compaction and session resets are frequent.

This paper documents five specific failure modes observed across OpenClaw versions 2026.1.24 through 2026.3.2, presents a three-layer fault-tolerant architecture that eliminated them in our deployment, and provides complete, reproducible deployment instructions. The architecture was developed and tested during a production deployment of an autonomous coding agent on a Mac Mini (M4, 24GB RAM) running macOS, at Speedrun AI Labs, Sydney. The agent model used was from the Anthropic Claude Sonnet family, released in the first quarter of 2026.

Our goal with this publication is to help other OpenClaw users who encounter these same failures. The open-source community benefits when practitioners document problems and solutions with specificity. This paper is written so that any user can feed it to their preferred AI assistant and receive actionable guidance for their own deployment.

2. Documented Failure Modes

We catalogued five distinct failure modes from OpenClaw's public issue tracker. Crucially, four of these five issues predate the project's governance transition in February 2026, indicating that memory system fragility was present during peak creator-led development. This is not a post-transition problem. It is a structural limitation of the default memory architecture that existed from early adoption.

Issue	Date Filed	Version	Description	Timing
#4868	Jan 30, 2026	2026.1.24	Memory index becomes dirty after updates; search returns empty results	Pre-transition
#8921	Feb 4, 2026	2026.2.x	Non-core memory plugins reported as "unavailable" in status command	Pre-transition
#9888	Feb 5, 2026	2026.2.x	memory-core plugin stops all persistence after updates (total breakdown)	Pre-transition
#15023	Feb 12, 2026	2026.2.x	MemorySearch fails with "database is not open" (SQLite lifecycle)	Pre-transition
#31677	Mar 2, 2026	2026.3.1	jiti v2.6.1 module resolver breaks SQLite native bindings (SQLITE_CANTOPEN)	Post-transition

Table 1: Documented memory system failures in OpenClaw, January to March 2026. All issues verified against github.com/openclaw/openclaw.

2.1 The Critical Regression: Issue #31677

The most impactful failure for production deployments was issue #31677, filed March 2, 2026 against version 2026.3.1. The regression was caused by an upstream dependency interaction: jiti v2.6.1 (a TypeScript/ESM runtime transformer maintained by the unjs project) resolves sqlite3 native bindings relative to its own module path instead of the sqlite3 package path. The actual binary file exists at the correct location (node_modules/sqlite3/build/Release/node_sqlite3.node), but jiti's resolution mechanism searches incorrect directories [4].

This is a known class of problem in jiti's issue tracker (issues #204, #346, #109). The regression manifests as a repeating SQLITE_CANTOPEN error that crashes the OpenClaw gateway every 2 to 5 minutes, causing a restart loop. Each restart re-initialises memory plugins, triggers another SQLite access attempt, and crashes again. The agent becomes unresponsive on all connected messaging channels during crash cycles.

This regression was triggered by a dependency interaction in the jiti module loader, not by an OpenClaw code change. Version 2026.2.26 (the last pre-regression release) used an earlier jiti version where native module resolution functioned correctly.

3. Three-Layer Fault-Tolerant Architecture

Our architecture addresses these failures through three independent, complementary systems. Each layer operates without dependency on the others, providing graceful degradation if any single component fails.

Layer	System	Function	Failure Mode
1 (Write)	Memory Flush	Writes durable notes to markdown before context compaction	Flush skipped if workspace is read-only
2 (Search)	QMD v1.1.6	BM25 + vector + LLM reranking on local markdown files	Falls back to keyword-only if models not downloaded
3 (Persist)	Mem0 v1.0.5 on Qdrant v1.13.0	Auto-recall before each turn; auto-capture after each turn	Falls back silently if Qdrant or embedding API down

Table 2: Three-layer fault-tolerant memory architecture.

3.1 Layer 1: Pre-Compaction Memory Flush

When a session approaches its context window limit, OpenClaw triggers compaction, which summarises and discards older messages. Without intervention, important context is permanently lost. The memory flush mechanism triggers a silent agentic turn before compaction, instructing the model to write durable notes to daily markdown files (memory/YYYY-MM-DD.md). This is configured via agents.defaults.compaction.memoryFlush in openclaw.json with a custom system prompt that prioritises architecture decisions, user feedback, blockers, and project status [5].

3.2 Layer 2: QMD Local Search

QMD (Query Markup Documents), created by Tobi Lutke, is a local-first search engine that combines BM25 full-text search, vector semantic search using local GGUF embedding models, and LLM-based reranking. Setting memory.backend = "qmd" in the OpenClaw configuration replaces the broken SQLite-based indexer entirely. QMD uses its own SQLite installation (via Homebrew on macOS), completely independent of the broken OpenClaw SQLite bindings. It indexes MEMORY.md and all daily log files under the memory/ directory [6][7].

3.3 Layer 3: Mem0 Self-Hosted on Qdrant

Mem0 is a memory engine that provides auto-recall (injecting relevant memories before each agent turn) and auto-capture (storing key facts after each turn). In self-hosted mode, it uses Qdrant as its vector store, completely bypassing OpenClaw's broken SQLite. We use the tensakulabs/openclaw-mem0 fork (v1.0.5), which fixes critical bugs in the official plugin: auto-recall silently discarding memories due to an incorrect property name (mem0ai/mem0#4037), embeddings ignoring configured baseURL (mem0ai/mem0#4040), and eager SDK imports crashing at startup [8].

The embedding API can be any OpenAI-compatible endpoint. Options include OpenAI directly, Azure OpenAI, Mistral, Voyage AI, local Ollama instances, or third-party routing services. The architecture is provider-agnostic by design.

3.4 Threats to Validity

These results may not generalise to non-macOS, non-local deployments or future OpenClaw versions. The architecture was tested on a single hardware configuration (Mac Mini M4, 24GB RAM) with a single agent

model family. Performance characteristics, particularly Mem0 auto-recall latency and QMD indexing speed, may differ on other platforms. Additionally, the three-layer architecture has been validated with dozens to low hundreds of stored memories; behaviour at larger scales has not been tested.

4. Configuration and Deployment

SECURITY NOTE: All API keys, tokens, and credentials in this paper are placeholder values. Never publish real credentials in configuration files, repositories, or documentation. Replace all instances of YOUR_API_KEY, YOUR_EMBEDDING_BASE_URL, and similar placeholders with actual values stored securely using environment variables, macOS Keychain, or OpenClaw's SecretRef mechanism. Commands in this section assume macOS with Homebrew, Docker Desktop, and Bun installed; adapt paths and package managers for Linux/Windows.

4.1 Disabling memory-core (Eliminating SQLITE_CANTOPEN)

The critical first step is disabling the memory-core plugin, which is the source of the SQLITE_CANTOPEN crash loop in OpenClaw 2026.3.x. Set plugins.slots.memory to "none" in openclaw.json:

```
"plugins": {
  "slots": {
    "memory": "none"
  }
}
```

Setting the slot to "none" disables memory-core and its SQLite dependency. The Mem0 plugin continues to operate through lifecycle hooks (before_agent_start and agent_end) independent of the slot system. The OpenClaw status command will report Mem0 as "disabled (memory slot disabled)." This is a known UX bug (issue #8921) and does not reflect actual plugin state. Verify operation through gateway logs, which will show: openclaw-mem0: initialized (mode: open-source, user: [userId], autoRecall: true, autoCapture: true).

IMPORTANT: Do not set plugins.slots.memory to "openclaw-mem0". While this appears logical, assigning any plugin to the memory slot triggers OpenClaw's built-in SQLite indexer alongside the plugin, reintroducing the SQLITE_CANTOPEN crash. The correct approach is "none" for the slot, with Mem0 operating through lifecycle hooks only. Minimum tested OpenClaw version: 2026.3.2; later versions may render workarounds obsolete.

4.2 Layer 1: Memory Flush Configuration

```
"agents": {
  "defaults": {
    "compaction": {
      "reserveTokensFloor": 40000,
      "memoryFlush": {
        "enabled": true,
        "softThresholdTokens": 4000,
        "systemPrompt": "Session nearing compaction.
          Extract only what is worth remembering.",
        "prompt": "Write durable notes to
          memory/YYYY-MM-DD.md. Focus on:
          architecture decisions, user feedback,
          blockers, code patterns, project status.
          Reply NO_REPLY if nothing worth storing."
      }
    }
  }
}
```

4.3 Layer 2: QMD Backend

Prerequisites: QMD installed via source build (github.com/tobi/qmd), SQLite installed via Homebrew (brew install sqlite), Bun runtime (bun.sh). Add to openclaw.json:

```
"memory": {
  "backend": "qmd"
}
```

QMD auto-downloads three GGUF models (approximately 2GB total) on first use for embedding, reranking, and query expansion. No API keys required. Runs entirely locally.

4.4 Layer 3: Mem0 Self-Hosted on Qdrant

Prerequisites: Docker running with Qdrant v1.13.0 container, tensakulabs/openclaw-mem0 v1.0.5 cloned to ~/.openclaw/extensions/openclaw-mem0, and access to any OpenAI-compatible embedding API. Start Qdrant:

```
docker run -d --name qdrant \
  -p 6333:6333 -p 6334:6334 \
  -v ~/qdrant-data:/qdrant/storage \
  --restart unless-stopped \
  qdrant/qdrant:v1.13.0
```

Configure Mem0 in openclaw.json. The example below uses openai/text-embedding-3-small as the embedding model. You can substitute any OpenAI-compatible embedding model. Replace all placeholder values with your actual credentials:

```
"plugins": {
  "entries": {
    "openclaw-mem0": {
      "enabled": true,
      "config": {
        "mode": "open-source",
        "userId": "your-agent-id",
        "autoCapture": true,
        "autoRecall": true,
        "oss": {
          "embedder": {
            "provider": "openai",
            "config": {
              "apiKey": "YOUR_API_KEY",
              "baseURL": "YOUR_EMBEDDING_BASE_URL",
              "model": "openai/text-embedding-3-small"
            }
          },
          "vectorStore": {
            "provider": "qdrant",
            "config": {
              "host": "localhost",
              "port": 6333,
              "collectionName": "agent-memories"
            }
          },
          "llm": {
            "provider": "openai",
            "config": {
              "apiKey": "YOUR_API_KEY",
              "baseURL": "YOUR_LLM_BASE_URL",
              "model": "YOUR_LLM_MODEL"
```

```
            }
          }
        }
      }
    }
  },
  "load": {
    "paths": [
      "~/.openclaw/extensions/openclaw-mem0"
    ]
  }
}
```

5. Verification

Verification proceeds in three phases, designed to abort safely if any step fails.

5.1 Phase 1: Diagnostic (Complete Before Proceeding)

Before applying the full configuration, verify that disabling memory-core eliminates the SQLITE_CANTOPEN error. Set plugins.slots.memory to "none", restart the gateway with "openclaw gateway restart", and monitor logs with "openclaw logs --follow" for 30 seconds. If SQLITE_CANTOPEN errors persist, the bug has a deeper source than the memory-core plugin and this architecture requires modification. In our testing on OpenClaw 2026.3.2 (build 85377a2), setting the slot to "none" immediately and completely eliminated all SQLite errors.

5.2 Phase 2: Component Verification

Component	Expected Log Entry	Indicates
Mem0	openclaw-mem0: initialized (mode: open-source, autoRecall: true, autoCapture: true)	Plugin loaded via lifecycle hooks
QMD	qmd memory startup initialization armed for agent "main"	QMD indexed workspace markdown files
Messaging	starting provider (@YourBot)	Channel connected
Gateway	Stable PID (no restart for 60 or more seconds)	No crash cycle; memory-core dead

Table 3: Component verification checkpoints.

5.3 Phase 3: Memory Persistence Test

The definitive test is cross-session memory persistence via your messaging channel: Step 1: Send /new (starts a fresh session). Step 2: Tell the agent a unique fact (for example, "My name is [Name]. I work at [Company]."). Step 3: Wait for the agent to acknowledge. Step 4: Send /new (starts another fresh session). Step 5: Ask the agent to recall the fact (for example, "What is my name?"). If the agent correctly recalls the fact across session boundaries, Mem0 auto-capture and auto-recall are functioning. In our production deployment, the agent recalled the user's full name, company affiliations, and role across session resets within seconds of the new session starting.

6. Stress Test Scenarios

Scenario 1: Context compaction during long session

Result: Memory flush writes to markdown before compaction. QMD indexes the new file. Mem0 auto-capture stores key facts in Qdrant during the session. After compaction, Mem0 auto-recall injects relevant memories on next message. Double coverage ensures no critical context is lost.

Scenario 2: Machine reboots (power outage or update)

Result: Docker restarts Qdrant automatically via the --restart unless-stopped flag. OpenClaw gateway restarts via LaunchAgent (macOS) or systemd (Linux). Mem0 reconnects to Qdrant with all memories

intact. QMD reindexes markdown files on boot. Full recovery without manual intervention.

Scenario 3: OpenClaw updates to future version

Result: memory-core is set to "none," so it is unaffected by SQLite binding changes in future releases. QMD is a separate binary, unaffected by OpenClaw updates. Mem0 plugin resides in the extensions folder, not managed by OpenClaw's update mechanism. Qdrant runs in Docker, completely isolated. The architecture is update-proof.

Scenario 4: Embedding API outage

Result: Mem0 auto-capture and auto-recall fail silently (by design). QMD continues working because it uses local GGUF models with no API dependency. Memory flush continues writing to markdown files. The agent keeps working without Mem0 recall until the API recovers. This is graceful degradation.

Scenario 5: Qdrant Docker container crashes

Result: Mem0 lifecycle hooks fail silently. QMD and memory flush continue normally. The --restart unless-stopped flag recovers Qdrant automatically. Data persists in the ~/qdrant-data/ volume mount. No data loss occurs.

7. Security Considerations

Publishing memory architecture details introduces potential attack surfaces. This section is descriptive, not prescriptive security advice; operators remain responsible for their own security posture. Apply the following measures to any deployment based on this paper:

Credential isolation: Never hardcode API keys in openclaw.json for production use. Use environment variables, macOS Keychain, or OpenClaw's SecretRef mechanism (available since version 2026.2.26), which supports 64 or more credential targets.

Network binding: Bind Qdrant to localhost only (127.0.0.1:6333). Never expose port 6333 to external networks. If remote access is needed, use Tailscale or SSH tunnels.

Qdrant authentication: Enable Qdrant API key authentication in production. The default Docker configuration has no authentication.

Memory content sensitivity: Mem0 stores extracted facts from conversations. If the agent handles sensitive data (financial, medical, personal), ensure Qdrant's storage directory (~/qdrant-data/) has appropriate filesystem permissions (chmod 700) and consider enabling FileVault disk encryption on macOS.

Messaging channel security: Configure messaging channel access controls (for example, Telegram's groupPolicy set to "allowlist" with authorised user IDs) to prevent unauthorised access to the agent.

Gateway binding: Bind the OpenClaw gateway to 127.0.0.1:18789 (loopback only), never 0.0.0.0. Use firewall rules to block LAN access if the machine is on a shared network.

Fork management: For production deployments, maintain a pinned fork of OpenClaw at a known-good version. Automatic updates can introduce regressions (as demonstrated by the 2026.3.1 SQLite breakage). Test updates on a staging agent before applying to production.

8. Historical Context: Open-Source Governance Transitions

OpenClaw's transition to foundation governance following its creator's move to OpenAI fits a documented pattern in open-source history. As of March 2026, OpenClaw has not exhibited the degradation seen in other cases (contributor counts grew, release velocity held, and security response remained rapid). The pattern is included here as context for why fault-tolerant architecture is prudent for any project undergoing governance change, not as a claim that degradation has occurred. We do not make any claim about OpenClaw's future trajectory; this is an argument for fault-tolerance in general.

Notable precedents include: MySQL forking to MariaDB after Oracle's Sun Microsystems acquisition (2010), with the original creator Monty Widenius leading the fork [9]; Redis relicensing from BSD to RSALv2/SSPLv1 in March 2024, causing all 12 non-employee external contributors to cease contributing, followed by the Linux Foundation launching Valkey within 8 days [10]; HashiCorp's BSL license change prompting the OpenTofu fork within 15 days, backed by 140 or more organisations [11]; and CentOS shifting to CentOS Stream after Red Hat/IBM's December 2020 announcement, spawning Rocky Linux and AlmaLinux within hours [12].

The CHAOSS project's 2024 quantitative analysis (Foster, arXiv:2411.04739) found that external contributors tend to drop significantly after restrictive relicensing, forks under neutral foundations achieve greater organisational diversity, and no evidence that relicensing improved vendors' financial results [10].

OpenClaw currently matches one commonly observed warning sign (creator departure to acquiring company) but exhibits several counter-indicators: continued MIT licensing, growing contributor base, rapid security response, and stated commitment to foundation governance. The fault-tolerant architecture presented in this paper is prudent regardless of future governance trajectory, as the memory system fragility existed independently of any transition.

9. Limitations and Future Work

Mem0 auto-recall latency: Each agent turn incurs an additional embedding API call for memory retrieval. On a Mac Mini with local Qdrant, this adds approximately 100 to 300 milliseconds per turn. For latency-sensitive applications, this may need tuning via the recallLimit and recallThreshold configuration parameters.

Memory volume scaling: The architecture has been tested with dozens to low hundreds of memories. Behaviour with thousands of stored memories (accumulated over months of use) has not been validated. Qdrant is designed for this scale, but embedding search quality may degrade without periodic memory curation.

QMD cold start: QMD's first run downloads approximately 2GB of GGUF models. This is a one-time cost but may be unexpected in bandwidth-constrained environments.

Status command UX bug: The "plugin disabled" warning from openclaw status (issue #8921) may cause confusion. Users should verify plugin state through gateway logs, not the status command.

Version dependency: This architecture was tested on OpenClaw 2026.3.2. Future versions may resolve the underlying SQLite binding issue, making the plugins.slots.memory = "none" workaround unnecessary. Conversely, future changes could introduce new incompatibilities. The time-sensitivity note at the beginning of this paper applies to all configuration details.

10. Responsible Disclosure

The bugs documented in this paper were reported through OpenClaw's public issue tracker on GitHub, which is the project's designated channel for bug reports. Issue #31677 (the SQLite/jiti regression) was filed by a community member on March 2, 2026 and remains open as of this paper's publication date. Issues #4868, #8921, #9888, and #15023 were similarly filed through the public tracker by various community members. This paper does not disclose any previously unreported vulnerabilities. All information presented is derived from publicly accessible sources: the OpenClaw GitHub repository, official documentation, and published blog posts.

11. Conclusion

OpenClaw's memory system exhibited recurring reliability failures throughout its rapid growth period from January to March 2026. The three-layer fault-tolerant architecture presented in this paper, combining pre-compaction memory flushing, QMD local hybrid search, and Mem0 self-hosted on Qdrant, eliminated these failures in our deployment while providing graceful degradation when individual components are unavailable.

The architecture was validated in a production deployment on OpenClaw 2026.3.2 (build 85377a2) running on a Mac Mini (M4, 24GB RAM). Memory persists across session boundaries, survives gateway restarts, and operates without dependency on OpenClaw's broken SQLite bindings. The total additional infrastructure cost is zero for search (QMD runs locally) and minimal for embeddings (standard API pricing for the embedding model of your choice).

This paper was written to help. Speedrun AI Labs believes in creating more value than we extract. If you are an OpenClaw user encountering these same memory failures, we hope this architecture saves you the

debugging time it cost us. Feed this paper to your preferred AI assistant and it should be able to guide you through the deployment.

References

[1] OpenClaw GitHub Repository. github.com/openclaw/openclaw. Accessed March 12, 2026.

[2] P. Steinberger, "OpenClaw, OpenAI and the future," steipete.me/posts/2026/openclaw, February 14, 2026.

[3] OpenClaw Release Notes, versions 2026.2.26 through 2026.3.8. github.com/openclaw/openclaw/releases. Accessed March 12, 2026.

[4] "mem0/SQLite bindings fail after 2026.3.1 update," Issue #31677, github.com/openclaw/openclaw/issues/31677. Filed March 2, 2026.

[5] OpenClaw Memory Documentation. docs.openclaw.ai/concepts/memory. Accessed March 11, 2026.

[6] T. Lutke, QMD (Query Markup Documents). github.com/tobi/qmd. Version 1.1.6.

[7] J. Casanova, "How to Fix OpenClaw's Memory Search with QMD," josecasanova.com/blog/openclaw-qmd-memory, March 10, 2026.

[8] tensakulabs/openclaw-mem0: Long-term memory plugin for OpenClaw agents. github.com/tensakulabs/openclaw-mem0. Version 1.0.5.

[9] MariaDB Foundation. mariadb.org. Fork initiated 2009 by M. Widenius; adopted by Fedora, OpenBSD, Arch Linux, Google, Wikimedia Foundation.

[10] D. Foster, "The New Dynamics of Open Source: Relicensing, Forks, and Community Impact," arXiv:2411.04739, November 2024.

[11] OpenTofu Manifesto and Linux Foundation announcement. opentofu.org. Fork announced August 25, 2023; production release v1.6.0 January 2024.

[12] Rocky Linux announcement by G. Kurtzer, December 2020. rockylinux.org.

[13] "memory-core plugin stops all persistence after updates," Issue #9888, github.com/openclaw/openclaw/issues/9888. Filed February 5, 2026.

[14] "Memory index becomes dirty after updates," Issue #4868, github.com/openclaw/openclaw/issues/4868. Filed January 30, 2026.

[15] "MemorySearch fails with 'database is not open'," Issue #15023, github.com/openclaw/openclaw/issues/15023. Filed February 12, 2026.

[16] "Non-core memory plugins reported as unavailable," Issue #8921, github.com/openclaw/openclaw/issues/8921. Filed February 4, 2026.

Appendix A: Version Matrix

Component	Version	Source
OpenClaw	2026.3.2 (build 85377a2)	npm: openclaw@2026.3.2
Node.js	22.x (LTS)	nodejs.org
QMD	1.1.6 (build 032f26e)	github.com/tobi/qmd
Mem0 Plugin	1.0.5 (tensakulabs fork)	github.com/tensakulabs/openclaw-mem0
Qdrant	1.13.0	Docker: qdrant/qdrant:v1.13.0
macOS	macOS (Apple Silicon, arm64)	Mac Mini M4, 24GB RAM
Docker	Docker Desktop for Mac	docker.com
Bun	1.3.10	bun.sh
Agent Model	Anthropic Claude Sonnet family (Q1 2026 release)	Via OpenAI-compatible endpoint

Appendix B: Step-by-Step Quick Reference

This appendix provides the complete fix in sequential order. You can feed this section to any AI assistant for guided deployment. All steps assume OpenClaw 2026.3.x is already installed and running. Commands assume macOS with Homebrew, Docker Desktop, and Bun installed; adapt paths and package managers for Linux/Windows.

Prerequisites (install these first):

```
# 1. Install QMD from source
git clone https://github.com/tobi/qmd ~/qmd-install
cd ~/qmd-install && bun install
# Create wrapper script at /usr/local/bin/qmd
# that calls: bun ~/qmd-install/src/qmd.ts "$@"

# 2. Install SQLite (for QMD)
brew install sqlite

# 3. Start Qdrant in Docker
docker run -d --name qdrant \
  -p 6333:6333 -p 6334:6334 \
  -v ~/qdrant-data:/qdrant/storage \
  --restart unless-stopped \
  qdrant/qdrant:v1.13.0

# 4. Install Mem0 plugin (tensakulabs fork)
git clone https://github.com/tensakulabs/\
  openclaw-mem0 \
  ~/.openclaw/extensions/openclaw-mem0
cd ~/.openclaw/extensions/openclaw-mem0
npm install
```

Configuration changes to ~/.openclaw/openclaw.json:

```
// CHANGE 1: Kill memory-core
// (stops SQLITE_CANTOPEN crash loop)
"plugins.slots.memory": "none"

// CHANGE 2: Enable QMD search backend
"memory.backend": "qmd"

// CHANGE 3: Enable memory flush
"agents.defaults.compaction.memoryFlush.enabled": true

// CHANGE 4: Enable Mem0 plugin
"plugins.entries.openclaw-mem0.enabled": true
// (with full oss config as shown in Section 4.4)

// CHANGE 5: Add plugin load path
"plugins.load.paths": [
  "~/.openclaw/extensions/openclaw-mem0"
]
```

Restart and verify:

```
# Restart the gateway
openclaw gateway restart

# Wait 20-30 seconds, then check logs
```

```
openclaw logs 2>&1 | tail -25

# You should see:
# - openclaw-mem0: initialized (...)
# - qmd memory startup initialization armed
# - No SQLITE_CANTOPEN errors
# - Stable gateway PID (no restart loop)

# Test memory persistence:
# 1. Send /new on Telegram
# 2. Tell agent: "My name is [your name]"
# 3. Send /new again
# 4. Ask: "What is my name?"
# If the agent remembers, you are done.
```

Troubleshooting:

```
# If SQLITE_CANTOPEN still appears:
# Verify plugins.slots.memory is "none"
# (not "openclaw-mem0" or "memory-core")

# If Mem0 shows as "disabled" in status:
# This is a known UX bug (issue #8921)
# Check logs instead: look for
# "openclaw-mem0: initialized"

# If agent does not respond on Telegram:
# Run: openclaw logs --follow
# Send a message and watch for errors

# If Qdrant is not running:
# docker start qdrant
# Wait 15 seconds, then restart gateway
```