

Agentic Memory for OpenClaw Builders

A practical structure for memory that stays useful under load.

Tag: #GrepTard

Audience: 15Grepples / OpenClaw builders

Date: 2026-04-06

Executive Summary

If you are building an agent and asking “how should I structure memory?”, the shortest good answer is this:

Do not build one giant memory blob.

Split memory into layers with different lifetimes, different write rules, and different retrieval paths. Most memory systems become sludge because they mix live context, task scratchpad, durable facts, and long-term procedures into one bucket.

A clean system uses:

- working memory
- session memory
- durable memory
- procedural memory
- artifact memory

And it follows one hard rule:

Retrieval before generation.

If the agent can look something up in a verified artifact, it should do that before it improvises.

The Five Layers

1. Working Memory

This is what the agent is actively holding right now.

Examples:

- current user prompt
- current file under edit
- last tool output
- last few conversation turns
- current objective and acceptance criteria

Properties:

- small
- hot
- disposable
- aggressively pruned

Failure mode:

If working memory gets too large, the agent starts treating noise as priority and loses the thread.

2. Session Memory

This is what happened during the current task or run.

Examples:

- issue number
- branch name
- commands already tried
- errors encountered
- decisions made during the run
- files already inspected

Properties:

- persists across turns inside the task
- should compact periodically
- should die when the task dies unless something deserves promotion

Failure mode:

If session memory is not compacted, every task drags a dead backpack of irrelevant state.

3. Durable Memory

This is what the system should remember across sessions.

Examples:

- user preferences
- stable machine facts
- repo conventions
- important credentials paths
- identity/role relationships
- recurring operator instructions

Properties:

- sparse
- curated
- stable
- high-value only

Failure mode:

If you write too much into durable memory, retrieval quality collapses. The agent starts remembering trivia instead of truth.

4. Procedural Memory

This is “how to do things.”

Examples:

- deployment playbooks
- debugging workflows
- recovery runbooks
- test procedures

- standard triage patterns

Properties:

- reusable
- highly structured
- often better as markdown skills or scripts than embeddings

Failure mode:

A weak system stores facts but forgets how to work. It knows things but cannot repeat success.

5. Artifact Memory

This is the memory outside the model.

Examples:

- issues
- pull requests
- docs
- logs
- transcripts
- databases
- config files
- code

This is the most important category because it is often the most truthful.

If your agent ignores artifact memory and tries to “remember” everything in model context, it will eventually hallucinate operational facts.

Repos are memory.

Logs are memory.

Gitea is memory.

Files are memory.

A Good Write Policy

Before writing memory, ask:

- Will this matter later?
- Is it stable?
- Is it specific?
- Can it be verified?
- Does it belong in durable memory, or only in session scratchpad?

A good agent writes less than a naive one.

The difference is quality, not quantity.

A Good Retrieval Order

When a new task arrives:

1. check durable memory
2. check task/session state
3. retrieve relevant artifacts
4. retrieve procedures/skills
5. only then generate free-form reasoning

That order matters.

A lot of systems do it backwards:

- think first
- search later
- rationalize the mismatch

That is how you get fluent nonsense.

Recommended Data Shape

If you want a practical implementation, use this split:

A. Exact State Store

Use JSON or SQLite for:

- current task state
- issue/branch associations
- event IDs
- status flags
- dedupe keys
- replay protection

This is for things that must be exact.

B. Human-Readable Knowledge Store

Use markdown, docs, and issues for:

- runbooks
- KT docs
- architecture decisions
- user-facing reports
- operating doctrine

This is for things humans and agents both need to read.

C. Search Index

Use full-text search for:

- logs
- transcripts
- notes
- issue bodies
- docs

This is for fast retrieval of exact phrases and operational facts.

D. Embedding Layer

Use embeddings only as a helper for:

- fuzzy recall

- similarity search
- thematic clustering
- long-tail discovery

Do not let embeddings become your only memory system.

Semantic search is useful.

It is not truth.

The Common Failure Modes

1. One Giant Vector Bucket

Everything gets embedded. Nothing gets filtered. Retrieval becomes mood-based instead of exact.

2. No Separation of Lifetimes

Temporary scratchpad gets treated like durable truth.

3. No Promotion Rules

Nothing decides what gets promoted from session memory into durable memory.

4. No Compaction

The system keeps dragging old state forward forever.

5. No Artifact Priority

The model trusts its own “memory” over the actual repo, issue tracker, logs, or config.

That last failure is the ugliest one.

A Better Mental Model

Think of memory as a city, not a lake.

- Working memory is the desk.
- Session memory is the room.
- Durable memory is the house.

- Procedural memory is the workshop.
- Artifact memory is the town archive.

Do not pour the whole town archive onto the desk.

Retrieve what matters.

Work.

Write back only what deserves to survive.

Why This Matters for OpenClaw

OpenClaw-style systems get useful quickly because they are flexible, channel-native, and easy to wire into real workflows.

But the risk is that state, routing, identity, and memory start to blur together.

That works at first. Then it becomes sludge.

The clean pattern is to separate:

- identity
- routing
- live task state
- durable memory
- reusable procedure
- artifact truth

This is also where Hermes quietly has the stronger pattern:

not all memory is the same, and not all truth belongs inside the model.

That does not mean “copy Hermes.”

It means steal the right lesson:

separate memory by role and by lifetime.

Minimum Viable Agentic Memory Stack

If you want the simplest version that is still respectable, build this:

1. small working context
2. session-state SQLite file
3. durable markdown notes + stable JSON facts
4. issue/doc/log retrieval before generation
5. skill/runbook store for recurring workflows
6. compaction at the end of every serious task

That already gets you most of the way there.

Final Recommendation

If you are unsure where to start, start here:

- Bucket 1: now
- Bucket 2: this task
- Bucket 3: durable facts
- Bucket 4: procedures
- Bucket 5: artifacts

Then add three rules:

- retrieval before generation
- promotion by filter, not by default
- compaction every cycle

That structure is simple enough to build and strong enough to scale.

Closing

The real goal of memory is not “remember more.”

It is:

- reduce rework
- preserve truth
- repeat successful behavior
- stay honest under load

A good memory system does not make the agent feel smart.

It makes the agent less likely to lie.

#GrepTard