READ MODE · ANSWER FIRST

How Hermes Agent Grows With You

An interactive, visual explanation of How Hermes Agent Grows With You, from setup through the complete system.

CHEAT SHEET · 6 ESSENTIAL IDEAS

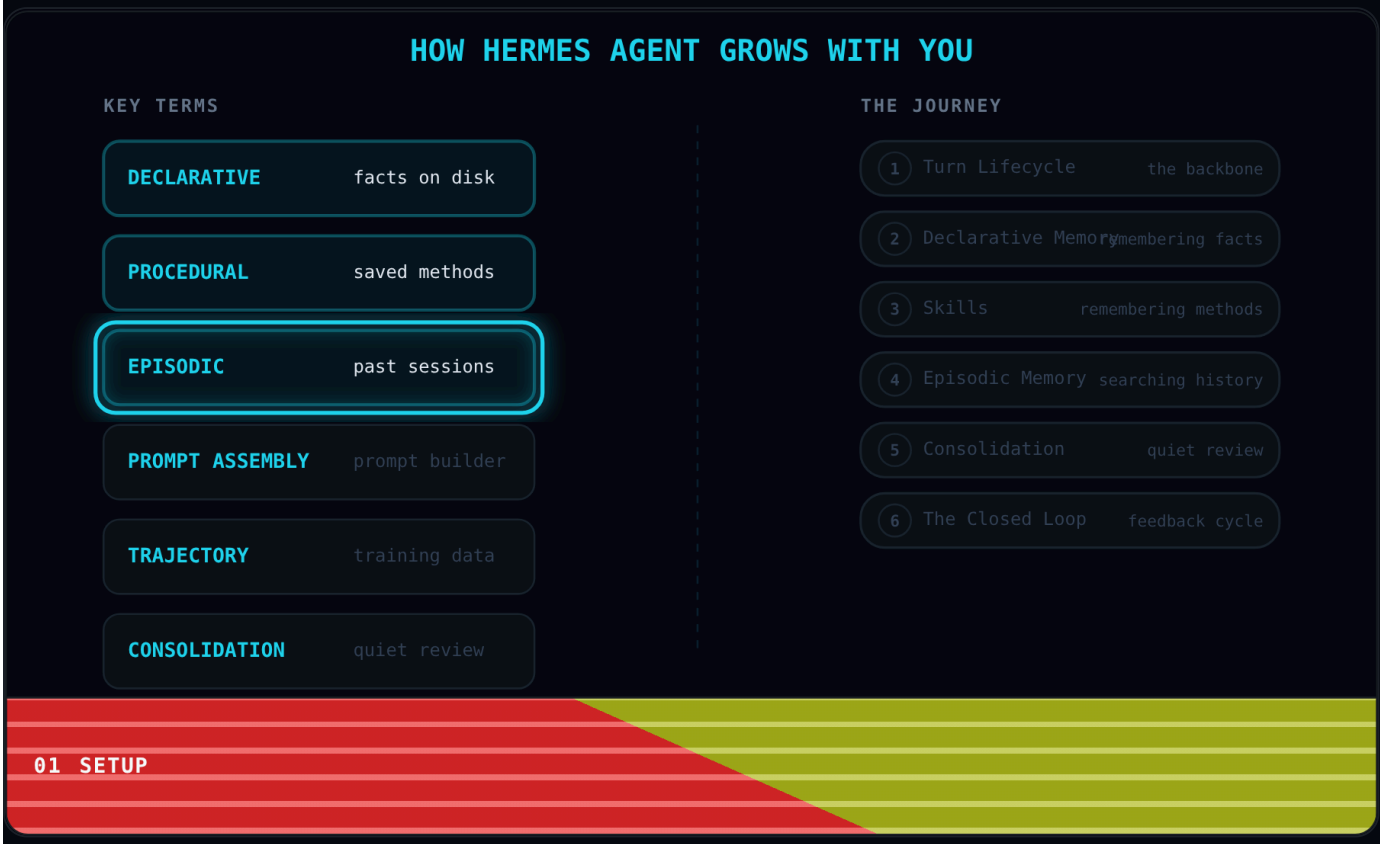
The whole story in 6 lines

How Hermes Agent Grows With You becomes easier to reason about when every stage is connected as one system.

1. The turn lifecycle is the backbone. Every learning mechanism plugs into this loop at specific boundaries.
2. Memory is bounded on purpose. The constraint forces the agent to curate and consolidate instead of hoarding.
3. Skills are procedural memory. They capture how to do recurring work, not just what is true about the environment.
4. Episodic memory is deliberately cold. It stays out of the prompt until explicitly retrieved, keeping costs bounded.
5. Background review separates the primary task path from the learning path. Answer first, learn second.
6. The learning loop is closed because artifacts from past work become inputs to future prompts. Solved work becomes future capability.

STAGE 1 OF 8

Setup



Welcome. Hermes Agent is a self-improving AI assistant. But self-improving does not mean the model rewrites its own brain during a chat. It means the system turns your conversations, corrections, and successful workflows into durable artifacts that change how it behaves next time. Let us learn how.

Declarative memory is the simplest form of learning. It stores compact facts about you and your environment in small text files. Think of it like a sticky note the agent keeps on its desk between conversations.

Procedural memory stores methods, not facts. When the agent solves a tricky multi-step task, it can save that workflow as a skill. Next time it faces the same kind of problem, it follows the recipe instead of figuring it out from scratch.

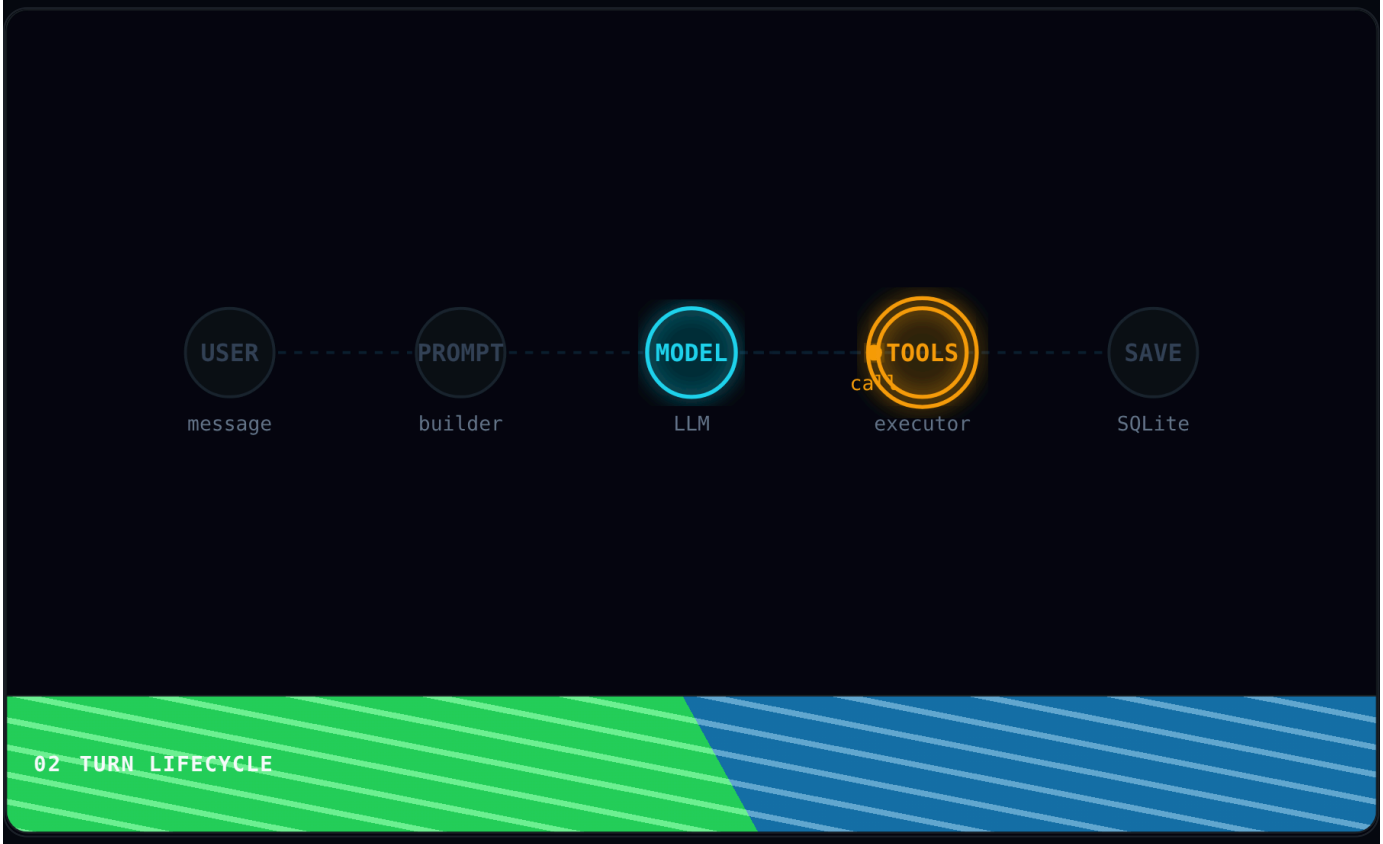
Episodic memory is the searchable archive of past conversations. Every session is saved to a database with full-text search. The agent does not carry all of history in its head. It searches when it needs to remember.

Prompt assembly is how learned knowledge re-enters the conversation. Facts, skill indexes, and recalled sessions are woven into the system prompt so the model can use them. A trajectory is the full record of a conversation exported for offline training.

Over the next six stages we will walk through the full lifecycle: how a single conversation turn works, how each memory type is created and reused, how the agent quietly reviews its own work, and how everything feeds back into a closed learning loop. Let us begin with a single turn.

STAGE 2 OF 8

Turn Lifecycle



The system is idle. The agent has its components wired and ready: prompt builder, model, tool executor, and session store. Every turn follows the same path through these components.

A user message arrives. Before the model sees it, the prompt builder assembles the system prompt. It pulls in frozen memory snapshots, the compact skills index, and any context files. This assembled prompt is the foundation the model reasons against.

The assembled prompt and user message reach the model. The model reads everything and decides what to do. It might answer directly, or it might call a tool first.

The model calls a tool. The agent executes the tool handler and feeds the result back. This loop can repeat multiple times. Try changing the Tool calls control in the sidebar to see what happens with zero or two tool calls.

The model produces its final response. The user sees the answer. But the turn is not over yet. Behind the scenes, the agent still has work to do.

The session is persisted. Every message, tool call, reasoning trace, and token count is saved to a SQLite database. This is the raw material that episodic memory and offline learning will use later. The turn is now complete.

STAGE 3 OF 8

Declarative Memory

We just saw how a turn flows through the agent. Now let us zoom into what happens when the agent notices a fact worth remembering. Declarative memory is the simplest learning path: compact, bounded, and always present in future prompts.

The agent identifies a stable fact during conversation. Maybe the user prefers dark mode. Maybe the staging server uses SSH port 2222. The agent decides this fact is durable, not temporary task progress, and initiates a memory write.

Before the fact is stored, it passes through a security scan. Hermes checks for prompt injection patterns, exfiltration attempts, and suspicious Unicode. Memory is reintroduced into future prompts, so it is treated as privileged input.

The fact passes capacity and duplicate checks. MEMORY.md has a budget of about 2,200 characters. USER.md gets about 1,375. If the store is full, the agent must consolidate or replace stale facts. Switch the Target control to USER to see the other file.

The fact is written atomically to disk. Hermes uses file locking and writes through a temporary file before committing with an atomic rename. Readers always see either the old complete file or the new complete file, never a partial write.

Here is the subtle part. The saved fact is durable on disk immediately, but it does not change the current session prompt. The prompt uses a frozen snapshot taken at session start. The new fact appears in the next session. This preserves prompt cache stability, which keeps costs predictable.

STAGE 4 OF 8

Skills

We now know how Hermes stores facts. But facts only capture what is true. Skills capture how to do things. When the agent solves a non-trivial task through trial and error, that successful workflow can become a reusable asset.

The agent completes a complex multi-step task. Perhaps it deployed a service to a staging VM, working through SSH quirks, container runtime differences, and config rendering. The solution involved multiple tools and several corrections.

The agent or a background reviewer decides this workflow deserves preservation. It creates a skill directory containing a SKILL.md file with the procedure, and optionally reference files, templates, or helper scripts.

The new skill is security scanned with the same rigor applied to community hub imports. Skills are executable knowledge. They influence future commands and file writes, so they are treated as privileged assets. Switch to Patch to see how existing skills are maintained.

The skill index cache is cleared. On the next prompt build, the new skill appears in the compact available skills block. The full skill text is not in the prompt. Only the metadata is. The model loads the full procedure on demand with skill_view when it recognizes a matching task.

A later session encounters a similar task. The model sees the skill in the index, loads it, and follows the procedure. One successful improvisation has become a repeatable capability. If the environment drifts, the agent uses fuzzy-match patching to update the skill incrementally.

STAGE 5 OF 8

Episodic Memory

05 EPISODIC MEMORY

We have seen facts in memory files and methods in skills. But what about the rich detail of a full troubleshooting session, or the reasoning behind a decision made three weeks ago? That is what episodic memory is for.

Every conversation is persisted to SQLite. Not just the user-facing text, but tool calls, reasoning traces, token counts, timestamps, and session lineage. An FTS5 full-text search index is kept synchronized by database triggers.

Historical sessions are too large and varied to live in the prompt. So episodic memory stays cold. The agent does not carry its entire history in context. It searches when a task calls for long-tail recall. Try switching the Query control to Broad to see how OR queries cast a wider net.

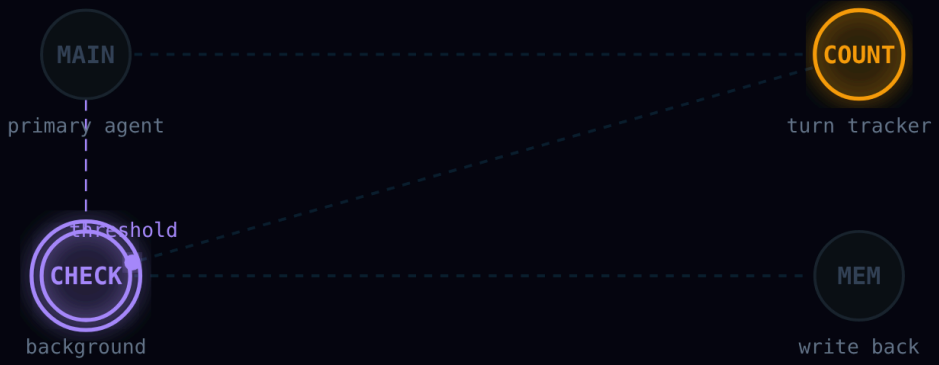
Retrieval is a two-stage pipeline. First, FTS5 finds matching messages ranked by relevance and groups them by session. Second, the top sessions are loaded, truncated around the most relevant spans, and summarized by an auxiliary model.

The returned artifact is an abstracted episodic memory, not a raw transcript dump. The summarization model distills the relevant sessions into a task-oriented recap. This keeps the main model focused on the current task while still benefiting from historical context.

The summarized recall is injected into the current conversation. The agent now has context it could not have fit in its prompt permanently. Episodic memory is the safety net that catches everything declarative memory and skills cannot hold. Now let us see how the agent decides what to save in the first place.

STAGE 6 OF 8

Consolidation



06 CONSOLIDATION

We have seen three types of knowledge the agent can save: facts, skills, and session archives. But when does it actually decide to save? It would be wasteful to stop mid-task and reflect after every sentence. Hermes solves this with background consolidation.

The main agent finishes its task and delivers the response to the user. The primary objective is always the user's problem. Learning is secondary. While the user reads the answer, the runtime checks its internal counters.

Hermes tracks two counters. One counts user turns since the last memory review. The other counts tool-calling iterations since the last skill review. When either counter crosses its threshold, a background review becomes eligible.

The runtime forks a background agent in a daemon thread. This reviewer is a separate AIAgent instance with the same model and tools but an isolated context. It receives the recent conversation and a review prompt. Try switching the Review control to see what it looks for.

The background reviewer examines the conversation. For memory review, it asks: did the user reveal preferences or facts worth remembering? For skill review, it asks: was a non-trivial approach used that required improvisation? It can call memory and skill tools directly.

If the reviewer finds something worth saving, it writes to memory or creates a skill. The user sees a brief notification. The counters reset. Compression boundaries provide another extraction opportunity: before context is summarized away, the agent gets one last chance to save durable facts.

STAGE 7 OF 8

The Closed Loop

07 THE CLOSED LOOP

We have walked through each knowledge form individually. Now let us see the complete picture. The power of Hermes is not in any single memory type. It is in how all four forms feed back into the prompt builder, creating a closed loop where past experience shapes future behavior.

Declarative memory provides the hot path. Facts from MEMORY.md and USER.md are loaded as a frozen snapshot and injected directly into the system prompt. These are always present, always available. They are the stable foundation.

The skills index provides the procedural path. A compact metadata listing of every saved skill is included in the prompt. When the model recognizes a matching task, it loads the full procedure on demand. Progressive disclosure keeps the prompt lean.

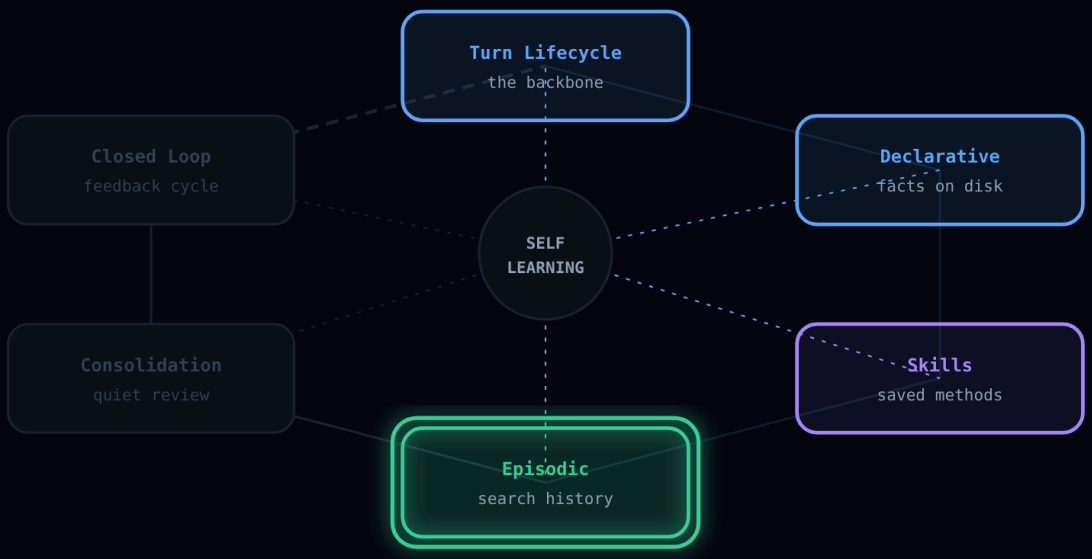
Episodic memory provides the cold recall path. Session search retrieves and summarizes relevant past conversations only when the model explicitly asks. Toggle the Ext. Memory control to see how external providers extend this recall.

Trajectories provide the offline path. Full conversation records are exported as JSONL for batch processing, evaluation, or reinforcement learning. This is the slow loop that can improve future models, not just future prompts.

All four paths converge in the prompt builder. A new session starts, and the agent is no longer blank. It knows your preferences, can follow procedures it learned, can search its own history, and its training data may include its best prior work. That is the closed loop. Solved work becomes future capability. Now let us step back and see the whole picture together.

STAGE 8 OF 8

Recap



08 RECAP

We started with the turn lifecycle: the backbone that everything plugs into. A user message enters, passes through prompt assembly, model reasoning, tool execution, and persistence. Every learning mechanism hooks into this loop at specific boundaries.

Then we learned how declarative memory works. Small, bounded text files that store facts about you and your environment. Scanned for security, written atomically, and injected as a frozen snapshot. The sticky notes on the agent's desk.

We saw how skills capture methods, not just facts. A successful multi-step workflow becomes a reusable procedure that future sessions can load and follow. The agent does not just remember what is true. It remembers how to do things.

Episodic memory gave the agent a searchable archive. Every conversation is persisted with full-text indexing. The agent does not carry its entire history in the prompt. It searches when it needs to remember, and a summarization model distills the results.

Background consolidation showed us the quiet learning pass. The agent delivers the answer first, then reviews its own work in a background thread. Counter-based triggers ensure periodic reflection without slowing down the primary task.

And the closed loop tied it all together. Memory, skills, session search, and trajectories all feed back into the prompt builder. Each conversation makes the next one a little smarter. That is what self-learning means in Hermes: solved work becomes future capability.