Give Claude a Memory That Survives Every New Chat

The exact setup so Claude stops starting from zero — and stops burning your tokens re-reading the same context.

Built by Axionox. I build AI systems for people for a living. This is the real thing I use, not a theory.

First, the problem nobody tells you about

Every new Claude chat starts from zero.

It doesn't matter that you explained your project yesterday. It doesn't matter that you already told it your stack, your goals, your weird preferences, the way you like things named. New session, blank slate. It forgot all of it.

So what do you do? You paste it all back in. The project summary, the last decisions, the "here's where we left off." Every single time.

That's the part that actually costs you. Every one of those re-paste chunks is tokens. You're paying — in money if you're on the API, in your usage cap if you're on a plan, and always in your own time — to teach the same thing over and over to something that never remembers.

It's like having a brilliant assistant with no short-term memory. Great for one conversation. Useless as a partner over weeks.

What it feels like when Claude actually remembers you

You open a fresh session. You type "keep going."

And it does. It knows what the project is. It knows the three decisions you made last week and *why* you made them. It knows you hate emoji in code comments and that you already ruled out the Postgres approach. You didn't paste anything. It just picked up where you left off.

That's the difference between a chat tool and a working partner. And you can set it up today, for free, in about ten minutes.

Here's exactly how.

The core idea: a "wrap-up" at the end of every session

Everything below is built on one simple pattern. Get this and the rest is just plumbing.

At the **end of each session**, you have Claude write a short summary of what happened — the decisions, the state of things, what's next — and save it to a store that lives *outside* the chat.

At the **start of the next session**, Claude reloads only that summary. Not the whole transcript. Just the compact recap.

That's it. Two moves: **save a summary out, read the summary back in.**

Why this beats pasting your whole history back:

- **It remembers across sessions** — the summary carries forward forever.
- **It uses far fewer tokens** — you're loading a half-page recap, not re-pasting an entire conversation. Cheaper every single time.

Think of the summary as Claude's journal. It writes the day's entry before bed and reads yesterday's entry when it wakes up.

Below are three ways to actually store that journal — from a real memory tool, to the NotebookLM "second brain" idea, to a version with zero setup. Pick one. They all use the same wrap-up pattern.

Path 1 — The memory MCP (recommended, official, reliable)

This is the one I'd start with. It's an official tool from Anthropic's own Model Context Protocol project, and it works in both Claude Desktop and Claude Code.

It gives Claude a **persistent knowledge-graph memory** — a structured store of facts, people, projects, and how they connect — that survives across every session automatically.

Repo: https://github.com/modelcontextprotocol/servers (it's the "memory" server)

Run it:

```
npx -y @modelcontextprotocol/server-memory
```

Wire it into Claude Desktop

Open your Claude Desktop config file:

- **Mac:** `~/Library/Application Support/Claude/claude_desktop_config.json`
- **Windows:** `%APPDATA%\Claude\claude_desktop_config.json`

Add the memory server:

```
{
  "mcpServers": {
    "memory": {
      "command": "npx",
      "args": ["-y", "@modelcontextprotocol/server-memory"]
    }
  }
}
```

Save, then fully quit and reopen Claude Desktop. You'll see the memory tools show up.

Wire it into Claude Code

One command:

```
claude mcp add memory -- npx -y @modelcontextprotocol/server-memory
```

Restart Claude Code and it's live.

The wrap-up, using this tool

At the end of a session, just say:

> "Save the key facts and decisions from this session to memory."

Claude writes them into the knowledge graph. Next session, at the start:

> "Load what you remember about this project from memory."

Now it's caught up. No pasting. Because it's a knowledge graph, it doesn't just store a blob of text — it stores *entities and relationships*, so it can connect "this client" to "that project" to "that decision" over time.

Tip: put those two lines into your project instructions (or your `CLAUDE.md` — see Path 3) so Claude does the wrap-up on its own without you asking.

Path 2 — The NotebookLM connector (optional, unofficial, honest caveat)

This is the "second brain" version — the one the reel was about. It's genuinely cool, and I want to be straight with you about it.

It connects Claude to your logged-in **NotebookLM** session. Claude can push a session summary into a notebook and query it back later. And because it's NotebookLM, you also get its real Studio outputs from your notes — **Audio Overviews, Video / Cinematic Overviews, infographics, and mind maps** — generated from the stuff Claude saved.

Repo: https://github.com/PleasePrompto/notebooklm-mcp

The honest caveat, up front: NotebookLM has no official public API. This connector works by driving your actual logged-in NotebookLM session in the browser. That means **it's unofficial and it can break whenever Google changes the NotebookLM interface.** It's not Google's product, and nobody can promise it'll keep working. Use it because it's genuinely useful — but don't build something mission-critical on it. If you want rock-solid, Path 1 is your answer.

With that said, here's the setup.

Install:

```
npx notebooklm-mcp@latest
```

Log in once — run its auth step so it can use your Google/NotebookLM session:

```
npx notebooklm-mcp@latest setup_auth
```

That opens a browser and asks you to log into Google one time. After that it reuses the session.

Add it to Claude Code:

```
claude mcp add notebooklm -- npx notebooklm-mcp@latest
```

The wrap-up, using NotebookLM

End of session:

> "Summarize this session and push it into my [notebook name] on NotebookLM."

Next session:

> "Query my [notebook name] for where we left off."

And the bonus — because it's NotebookLM — you can then ask for an **Audio Overview** or a **mind map** of your accumulated project notes and get a genuinely nice recap you can listen to on a walk. That part is real and it's the reason this path is worth trying even with the caveat.

Path 3 — The no-tools version (zero setup, works right now)

Don't want to install anything? You don't have to. If you use **Claude Code**, you already have a memory file built in: `CLAUDE.md`.

Claude Code automatically reads `CLAUDE.md` from your project at the start of every session. So it *is* the "read the summary back in" half of the pattern — for free, no tools, no config.

All you add is the wrap-up half. Here's the whole thing.

1. Make the file in your project root:

```
touch CLAUDE.md
```

2. Put a decision log section in it:

```
# Project Memory

## Decision Log
- (Claude appends short dated entries here at the end of each session)
```

3. Add the wrap-up instruction to that same file so Claude does it automatically:

```
## End-of-session wrap-up
At the end of each session, append a short entry to the Decision
Log above:
date, what we did, key decisions + why, and what's next. Keep it
to a few lines.
```

Now, at the end of a session, say:

> "Do the wrap-up."

Claude appends something like:

```
- 2026-07-05: Chose the memory MCP over NotebookLM for
reliability.
  Wired it into Claude Desktop. Next: test the auto-save on the
client project.
```

Next session, Claude reads `CLAUDE.md` on its own and it's already caught up. Same pattern as the fancy tools — save a summary out, read it back in — just done with one plain text file. Honestly, for a lot of people this is enough.

The bigger picture: memory is one piece of a real agent

Memory isn't a trick on its own. It's one of the parts that turns Claude from a chat window into something that actually *works alongside you* over time.

A real setup usually has a few of these:

- **Memory** — so it remembers across sessions (that's this guide).
- **Tools / MCP servers** — so it can *do* things: read your files, hit your database, drive an app.
- **Instructions** — a `CLAUDE.md` or system prompt so it knows your standards without being told.

Once memory is in place, everything else compounds, because the agent stops re-learning your world every morning. That's the whole point. You're not just saving tokens — you're building something that gets more useful the longer you use it, instead of resetting to zero.

Your next step

Pick one path and set it up in the next ten minutes. My honest order:

1. **Start with Path 1** (the memory MCP) — official, reliable, works in Desktop and Code. 2. **Add Path 2** (NotebookLM) if you want the second-brain feel and the audio/mind-map outputs — knowing it's unofficial and can break. 3. **Or just do Path 3** (`CLAUDE.md`) today if you want zero setup and you're in Claude Code.

Then add the two wrap-up lines to your project instructions so Claude saves and reloads on its own. Do it once and every future session starts warm instead of cold.

Built by Axionox. I build AI systems for a living — the real, working kind, not the demo kind. Follow for more setups like this one.