Q2 · 2026

CLAUDE.MD · HOOKS · SKILLS · SUBAGENTS · CHECKPOINTS · MCP

Memory.

Engineered context.
Not chat history.
Built to survive.

Not a tutorial. A playbook for engineers who ship every day with Claude Code — CLI and IDE.

THE BRIEF, IN THREE BEATS

01 Memory is **engineered** — a hierarchy of files you curate, not chat you hope survives.

02 Four layers: Enterprise · Project · User · Session — each with its own file, path, and job.

03 Hooks + checkpoints + MCP memory servers are the difference between L2 and L4.

SLIDE 02 · HOOK

"Claude forgot again."

— because you never **told it to remember.** Memory is a ***system***, not a hope.

*Every "explain the codebase again" is a memory bug. Every context you re-type is **1-3k tokens** Claude already had — if the right file existed. A 20-line CLAUDE.md pays for itself in the first session.*

Sources: code.claude.com/docs/en/memory · anthropic.com "How Anthropic teams use Claude Code" (PDF) · joseparreogarcia.substack.com Claude Code Memory Explained.

THE CLAIM

Memory in Claude Code isn't chat history. It's a file system you curate — every session boots from files, and the ones you wrote decide what Claude knows.

THREE COROLLARIES

#1 **The prompt is short-term. The files are long-term.** If it's not in a file, it dies at `/compact`.

#2 **Every file costs tokens.** A 500-line CLAUDE.md consumes ~4-8k tokens on *every* prompt. Curate ruthlessly.

#3 **Files rot faster than code.** Treat CLAUDE.md like a test — if it hasn't been touched this month, it's probably wrong.

THE HONEST TRADE

The trap. Kitchen-sink CLAUDE.md — everything you might ever want Claude to know. Grows to 800 lines. Nobody reads it. Token bloat eats the context window.

The move. Think in layers: enterprise policy, project rules, personal defaults, session notes. Each file owns one job.

The result. Claude picks up where you left off — because the files told it what actually matters.

Sources: code.claude.com/docs/en/memory · anthropic.com "How Anthropic teams use Claude Code" · agentfactory.panaversity.org CLAUDE.md configuration hierarchy.

THE FRAME

Four files. Four scopes. Higher layers override lower ones. All auto-loaded on session start.

LAYER	FILE PATH	WHO OWNS	GIT?	WHAT LIVES HERE
1 · Enterprise org-wide default	/Library/Application Support/ClaudeCode/CLAUDE.md (macOS · similar paths on Linux/Win)	IT / Security team. *Managed device only.*	No — deployed via MDM.	Security policies, forbidden tools, compliance guardrails.
2 · Project team-shared	./CLAUDE.md ./.claude/settings.json (repo root)	Team. Reviewed like code.	✓ committed	Architecture, coding standards, tests, "how we work".
3 · User your defaults	~/.claude/CLAUDE.md ~/.claude/settings.json (home directory)	You. Applies to *all* your projects.	✗ never	Personal style, editor, alias'd commands, tone.
4 · Session in-flight	~/.claude/projects/<hash>/ (session log — auto-managed)	Claude Code itself.	✗ never	Chat + tool calls. Restored via `--resume` / `--continue`.

TOKEN BUDGET · ROUGH NUMBERS

Enterprise ~500-1500 tok · Project ~1500-4000 tok · User ~500-1500 tok · Session grows unbounded until `/compact`.
Total memory tax: ~3-7k tokens per prompt. On 200k window: 1.5-3.5% "always spent".

LOAD ORDER · HIGHER WINS

`1 Enterprise → 2 Project → 3 User → 4 Session`
Session prompts override User. User overrides Project. Project overrides Enterprise *only where explicit.* Enterprise policies (bans, guardrails) are non-negotiable.

Sources: code.claude.com/docs/en/memory (4-tier hierarchy) · agentfactory.panaversity.org CLAUDE.md configuration hierarchy · github.com/luongnv89/claude-howto/02-memory.

SLIDE 05 · TIMELINE — HOW MEMORY GOT TO WHERE IT IS TODAY

FEB 2025 → Q2 2026

15 months, six shifts. Claude Code went from "chat with amnesia" to a persistent workspace.

WHEN	SHIP	WHAT IT UNLOCKED
Feb 2025 launch	**Claude Code launches with CLAUDE.md**	Single project-level file, auto-loaded on start. Establishes the pattern: *files are memory*.
Aug 2025 +6 months	**/memory command + hierarchy**	Multi-tier hierarchy (Enterprise / Project / User / Session). `/memory` lists all loaded files. # shortcut adds a line in-flight.
Late 2025 +9 months	**Hooks lifecycle (12 events)**	`SessionStart` · `PreCompact` · `UserPromptSubmit` · `PreToolUse` + 8 more. Memory becomes *programmable*.
Q1 2026 +11 months	**Skills · Subagents · Plugins**	Each subagent gets its own CLAUDE.md + tool scope + isolated context. Skills = reusable memory-triggers as folders with `SKILL.md`.
Jun 2026 +16 months	**Checkpointing · /rewind · ESC ESC**	Roll back *both* code and conversation. Auto-checkpoint before risky ops. Session persistence gets fixed ("amnesia fix").
Q2 2026 now	**MCP memory servers ecosystem**	`mcp-knowledge-graph` · `mem0-mcp-selfhosted` · `memory-graph`. Memory now scales *outside* the file system — knowledge graphs, vector stores, per-project notebooks.

Sources: code.claude.com/docs (memory, hooks, skills) · sitepoint.com Claude Code June 2026 features · hidekazu-konishi.com Claude Code hooks complete guide · awesomeclaude.ai MCP memory servers.

THE STRUCTURE

A production CLAUDE.md has 8 sections. Each earns its token cost — or gets cut.

SECTIONS 1-4 · ORIENTATION + RULES

§1 · Project one-liner. What this repo IS in one sentence. *"E-commerce checkout API. Node 20, Fastify, Postgres, Stripe."*

§2 · Architecture map. Directories that matter. What lives where. `src/`, `packages/`, `infra/` — and what's off-limits.

§3 · Coding standards. Style bullets Claude must follow. TypeScript strict, no any, no default exports, tests co-located.

§4 · Testing rules. How tests are run. What must pass before a commit. Which paths never ship without a test.

SECTIONS 5-8 · WORKFLOW + GUARDRAILS

§5 · Commands cheat-sheet. `npm run …`, migrations, generators, common one-liners. Save Claude 3 guesses per session.

§6 · Do / Don't for THIS repo. Explicit rules: *never touch migrations directly · always use zod at the boundary · Sentry stays in `src/lib/telemetry`.*

§7 · Current focus / WIP. One paragraph on *what we're building right now.* Updated weekly, not per-PR.

§8 · Imports. `@import ./docs/architecture.md` for deep dives. Keep the main file short; pull the rest on demand.

Target length: 100-200 lines · ~1500-4000 tokens · fits on one printed page. Anything longer, split via `@import`.

Sources: uxplanet.org CLAUDE.md best practices · anthropic.com "How Anthropic teams use Claude Code" · mcp.directory/blog/claude-code-best-practices · alexop.dev CLAUDE.md customization guide.

FIELD EVIDENCE

One is 500 lines of good intentions. The other is 120 lines that survived 6 months.

THE FAILURE · 500 LINES, 6 WEEKS OLD

Symptoms:

- The full `package.json` pasted inline. *~200 tokens Claude never reads.*
- "Team member David likes tabs over spaces" — 8 personal preferences that belong in `~/.claude/CLAUDE.md`.
- "We migrated to Postgres last quarter" — a fact, not a rule. Belongs in a design doc.
- Copy-pasted README + copy-pasted CONTRIBUTING.md. *Duplicated, drifted.*
- "TODO: fix this section" from March. Still there in July.

> **Token cost:** ~7500 tokens per prompt. Cache misses often — because the file changes weekly with noise.

THE SURVIVOR · 120 LINES, 6 MONTHS OLD

What kept it alive:

- Every section answers "what will Claude get wrong *tomorrow*?" — not "what could someone want to know?"
- No facts, only rules. "Use zod at the boundary" — not "we use zod because …"
- Deep dives are `@import ./docs/…` — loaded on demand, not on every prompt.
- Reviewed like code — every change goes through PR review.
- "Current focus" block updated *weekly*, deleted when the sprint ends.

> **Token cost:** ~1800 tokens. Cache-friendly — content stable. **4× cheaper** per prompt.

The 500-line rule: when your CLAUDE.md crosses 500 lines, it's a signal you're using it as a wiki. Split it into `@imports` or turn sections into skills.

Sources: uxplanet.org CLAUDE.md best practices · medium.com luongnv89 CLAUDE.md DNA · reddit r/Anthropic "best practices early 2026" · joseparreogarcia Claude Code Memory Explained.

LAYER 3 · YOU ONLY

The file nobody sees — where your personal defaults live, never committed, always loaded.

WHAT BELONGS HERE

✓ **Communication style.** "Be terse. No preamble. Skip disclaimers." Saves ~200 tokens per response.

✓ **Editor / shell defaults.** "I use nvim, tmux, fish. Prefer `rg` over grep, `fd` over find."

✓ **Universal language prefs.** "Always use pnpm, never npm. UV, not pip. Bun, not node — unless the repo dictates otherwise."

✓ **Personal safety rails.** "Never `rm -rf`. Ask before force-push. Confirm before dropping tables."

✓ **Common patterns you always use.** "Prefer functional over classes. Small files. Colocated tests."

> **Rule:** if it applies to *every* repo you touch, it lives here. If it's repo-specific, it lives in the project file.

WHAT DOES NOT BELONG

✗ **Secrets.** API keys, tokens, credentials. Ever. Use `.claude/settings.local.json` with env-var refs.

✗ **Repo-specific facts.** "This project uses Postgres" → belongs in the project's CLAUDE.md, not yours.

✗ **Team conventions.** "We do TDD" is a team rule; commit it.

✗ **Session context.** "Currently working on issue #123" — that's session-level, not personal-default.

✗ **Anything you'd want a colleague to know.** They can't read your ~/.claude/. If it's shared knowledge, it's the wrong file.

> **Length target:** 30-80 lines. Yours is a signature, not a manifesto.

Sources: code.claude.com/docs/en/memory · youngleaders.tech "How I Finally Sorted My Claude Code Memory" · agentfactory.panaversity.org CLAUDE.md configuration hierarchy.

One directory, six things. Know which one gets committed — and which one is *never* in git.

PATH	WHAT IT DOES	GIT?	TYPICAL SIZE
./CLAUDE.md project brain	The main memory file. 8 sections. Read on every session start. Team-shared.	✓ committed	100-200 lines
.claude/settings.json team config	Permissions · MCP servers · hook definitions · model defaults. Team-wide.	✓ committed	20-80 lines
.claude/settings.local.json your overrides	Your local tweaks. Secrets via env refs. Your API keys, your allowed commands.	✗ .gitignore'd	10-40 lines
.claude/commands/*.md slash commands	Reusable prompts triggered by /name. */review-pr*, */write-migration*, */refactor-tests*.	✓ committed	5-50 files
.claude/agents/*.md sub-agents	Specialised agents with own context + tool scope. YAML frontmatter + prompt body.	✓ committed	3-15 files
.claude/hooks/ lifecycle scripts	Shell scripts that fire on 12 lifecycle events. Deterministic guardrails + memory triggers.	✓ committed	2-10 scripts

Sources: hidekazu-konishi.com Claude Code features & settings reference 2026 · morphllm.com Claude Code hooks 2026 · alexop.dev CLAUDE.md vs Skills vs Subagents.

ISOLATION IS THE POINT

Each sub-agent gets a fresh context + its own file + its own tool scope. That's the feature.

.CLAUDE/AGENTS/REVIEWER.MD · ANATOMY

```
---
name: reviewer
description: Reviews PRs against project
  standards. Blocks merges that fail.
tools: Read, Grep, Bash(git log:*)
model: sonnet-5
---

You are a strict code reviewer. Read
the diff. Check against §3 of CLAUDE.md.
Block if: no tests · any() types · new
deps without justification.

Output format: PASS / FIX with line refs.
```

THE 3 RULES OF SUB-AGENT MEMORY

Rule 1 · Their context is fresh. They don't see your session's chat. They see *only* what the orchestrator hands them + their own file + shared CLAUDE.md.

Rule 2 · Their tools are scoped. Frontmatter `tools:` is a *whitelist*. Reviewer sees Read + Grep + `git log` — never Write, never `rm`. Least privilege.

Rule 3 · Their output is the memory. When a sub-agent finishes, only its *final response* returns to the orchestrator. Everything else is discarded. Design the output like an API contract.

> **Depth cap:** 5 levels of nesting (Fable 5 launch). Any deeper and coherence collapses.

Sources: totalum.app Claude Code subagents 2026 playbook · levelup.gitconnected mental model for Claude Code · alexop.dev CLAUDE.md vs Skills vs Subagents · github VoltAgent/awesome-claude-code-subagents.

RECURSIVE · DEPTH 5

Turn one monolithic file into a composable network. Small root, deep only when needed.

SYNTAX · @PATH/TO/FILE

```
# CLAUDE.md · root · 40 lines

## Project
E-commerce checkout API. Node 20.

## Architecture
@docs/architecture.md

## Coding standards
@.claude/rules/typescript.md
@.claude/rules/testing.md

## Current focus
Migrating Stripe → Stripe v3.
@.claude/wip/stripe-migration.md
```

THE 5 RULES OF GOOD IMPORTS

01 · **Root stays short.** Target 40-80 lines. Everything deep is behind an `@import`.

02 · **One @import per concept.** One file for architecture, one for style, one for testing. Not "misc.md".

03 · **Depth ≤ 5.** A → B → C → D → E. Deeper collapses; you're building a graph, not a tree.

04 · **Never @import the whole README.** The README is for humans. Distil rules; don't copy prose.

05 · **WIP goes in an @import.** "Current focus" lives in `.claude/wip/*.md` — deleted when the sprint ends.

> **Why:** Claude only loads what the current task needs. Cold @imports don't burn tokens until referenced.

Sources: code.claude.com/docs/en/memory (@import syntax) · claudefa.st Memory optimization guide · medium.com luongnv89 CLAUDE.md DNA (recursive imports, depth 5) · youtube CC memory 6-layer.

PARITY MATRIX

Same files. Same commands. Different UX — and one gotcha you must know.

MEMORY FEATURE	CLI (CLAUDE)	VS CODE EXT	JETBRAINS PLUGIN
CLAUDE.md hierarchy (4 tiers)	✓ full · auto-load	✓ full · auto-load	✓ full · auto-load
/memory · #shortcut · @import	✓ all supported	✓ all supported	✓ all supported
--resume / --continue	✓ picker + latest	✓ UI history pane	✓ UI history pane
/rewind · checkpointing (Jun 2026)	✓ /rewind + ESC ESC	✓ + timeline UI	✓ + timeline UI
Hooks (12 lifecycle events)	✓ all fire	✓ all fire	✓ all fire
MCP servers	✓ .claude/settings.json	✓ shared config	✓ shared config
Open editor tabs → context	✗ N/A — no editor	✓ auto-included	✓ auto-included
Session store location	~/.claude/projects/<hash>/	← same path	← same path

The gotcha: IDE sessions and CLI sessions share the same on-disk store — but starting a CLI session in the same repo *while* an IDE session is open can fork the checkpoint history. Rule: one live session per repo, per machine.

Sources: hidekazu-konishi.com Claude Code features 2026 reference · padiso.co Claude Code JetBrains vs VS Code · reddit r/ClaudeAI IDE extension · threads @boris_cherny resume feature.

ENTER · EXIT · RESUME

Every command changes what Claude remembers tomorrow. Know the tradeoffs.

COMMAND	WHAT IT DOES	WHAT SURVIVES	USE WHEN
`/compact` manual summary	Claude writes a summary of the session, drops the raw messages, keeps the summary + CLAUDE.md files.	The summary + all loaded files. Raw chat lost.	Context hitting 70-80% full and you're mid-task.
`/clear` hard reset	Wipes all chat history. No summary written. CLAUDE.md re-loads on next prompt.	Only your files. Everything else — gone.	Switching tasks completely. Fresh start.
`claude --resume` picker	Shows a list of past sessions for this repo. Pick one to resume — full chat, tool state, checkpoints restored.	Everything — chat, tool history, checkpoints.	Coming back after hours or days.
`claude --continue` latest	Auto-picks the most recent session for this repo. No prompt.	Everything from the last session.	Quick "back to what I was doing".
`/rewind · ESC ESC` Jun 2026 · checkpoint	Rolls back *both* code AND conversation to a checkpoint. Fixes bad edits without losing history.	The state as of the checkpoint.	Bad refactor · wrong direction · destructive op.

Sources: code.claude.com/docs (session management) · github luongnv89/claude-howto 08-checkpoints · mindstudio.ai /rewind command · explainx.ai ESC ESC checkpoint restore · sitepoint.com June 2026 features.

THE PRUNING EVENT

Auto-compaction fires around 85% context full. The `PreCompact` hook is your last save.

ANATOMY OF A COMPACTION

Trigger: Context ~85% full, or manual `/compact`.

What Claude does: Reads the whole conversation. Writes a summary (~2-4k tokens) that keeps decisions + file paths + open questions.

What survives:

- ✓ CLAUDE.md hierarchy (re-loaded)
- ✓ The compaction summary
- ✓ Currently-open files (IDE only)

What dies:

- ✗ Raw chat turns
- ✗ Tool-call transcripts (only their conclusions)
- ✗ Thinking blocks from earlier turns
- ✗ Anything you didn't tell it to remember

THE PRECOMPACT → SESSIONSTART PATTERN

The community's "pseudo-PostCompact" pattern — save state before, restore after:

```
// .claude/hooks/pre-compact.sh
echo "$CLAUDE_SESSION_ID" > \
   .claude/state/last-session.txt
cat conversation-tail.md > \
   .claude/state/pre-compact-notes.md
```

```
// .claude/hooks/session-start.sh
if [ -f .claude/state/pre-compact-notes.md ];
then cat .claude/state/pre-compact-notes.md; fi
```

Result: when Claude wakes up after compaction, it reads your *pre-compact save* as context injected on session start.

Sources: datacamp.com Claude Code hooks · reddit r/ClaudeAI Pseudo-PostCompact pattern · mindstudio.ai post-compaction hooks · morphllm.com Claude Code hooks 2026.

4 OF 12 LIFECYCLE EVENTS

Hooks aren't just guardrails. They're deterministic memory triggers — code you run to shape context.

01 · SESSIONSTART

Inject context on every session

Use for: loading last-session notes, current branch, open TODOs, WIP-file diffs, sprint focus.

Example: `echo "Branch: $(git branch --show-current)"; cat .claude/state/wip.md`

02 · USERPROMPTSUBMIT

Fire on every user prompt

Use for: just-in-time context — inject the current file's tests, related migrations, or recent commits based on what the user just typed.

Also: redact secrets · block risky prompts · log the intent.

03 · PRECOMPACT

Save what's about to die

Use for: writing "decision log" for what was concluded this session, so `SessionStart` can restore it.

Example: append last N turns to `.claude/state/decisions.md`. Community pattern nicknamed *pseudo-PostCompact*.

04 · STOP & SUBAGENTSTOP

Close the loop, write the summary

Use for: write "what I did" to `.claude/state/session-log.md`. Commit-message drafts. Handoff notes.

SubagentStop: capture sub-agent findings before they're discarded — Fable 5 workflows lose everything otherwise.

Full list (12 events): `SessionStart` · `Setup` · `SessionEnd` · `UserPromptSubmit` · `UserPromptExpansion` · `PreToolUse` · `PostToolUse` · `PreCompact` · `Notification` · `Stop` · `SubagentStop` · `PrecompileHook`

Sources: morphllm.com Claude Code hooks 2026 (full event list) · hidekazu-konishi.com Claude Code hooks complete guide · claudefa.st hooks guide · github disler/claude-code-hooks-mastery · totalum.app hooks playbook.

MCP MEMORY SERVERS

Files don't scale to knowledge graphs. MCP memory servers do — use them when it earns its bill.

THE 3 SERVERS PROS ACTUALLY USE

mcp-knowledge-graph `(Shane Holloway)`
Local knowledge graph. Persistent across sessions. Best for capturing *relationships* — "X depends on Y", "team decided Z on date D".

mem0-mcp-selfhosted `(community fork)`
Long-term memory backed by your infra. Vector-search over past sessions + user prefs. Ideal when the same team hits the same codebases repeatedly.

memory-graph `(Gregory Dickson)`
Code-focused. Indexes patterns + relationships in your codebase. Neo4j-compatible under the hood.

FILES VS MCP · WHEN TO ESCALATE

Stay in files when:

- You have < 10 recurring context items.
- Team fits in one repo.
- Rules matter more than facts.

Reach for MCP memory when:

- Cross-repo knowledge (monorepo of monorepos).
- Relationships matter: *who owns · what depends*.
- You've hit "CLAUDE.md is 800 lines" and splitting via @import isn't enough.
- Team > 10 people sharing memory.

> **The trap:** installing an MCP memory server before you exhausted CLAUDE.md + @imports. Complexity you don't need yet.

Sources: github shaneholloman/mcp-knowledge-graph · reddit r/ClaudeAI mem0-mcp-selfhosted · mcpservers.org gregorydickson/memory-graph · glama.ai knowledge-memory MCP directory · awesomeclaude.ai memory servers.

SYMPTOM · FIX · COST

If you recognise your workflow here — fix it this week.

AP1

Kitchen-sink CLAUDE.md

🔍 **Symptom:** 500+ lines, everyone adds, nobody removes.

💊 **Fix:** Split via `@import`. Root stays ≤ 200 lines.

⏱ **Cost:** ~6k extra tokens per prompt · cache misses when file changes.

AP2

Facts, not rules

🔍 **Symptom:** "We migrated to Postgres last quarter." History, not action.

💊 **Fix:** Rewrite as a rule. *"All new persistence goes through `src/db/`."*

⏱ **Cost:** Tokens spent on trivia · rots into "outdated" within weeks.

AP3

Secrets in memory files

🔍 **Symptom:** API keys · DB URLs · tokens pasted into CLAUDE.md for "convenience".

💊 **Fix:** `.claude/settings.local.json` (gitignored) with env-var refs. Never in the file that ships.

⏱ **Cost:** One PR leaks them all forever.

AP4

Stale WIP block

🔍 **Symptom:** "Currently working on Stripe migration" — from three months ago.

💊 **Fix:** WIP goes in `.claude/wip/*.md` — deleted when sprint ends. Weekly reviews.

⏱ **Cost:** Claude confidently follows outdated priorities.

AP5

Copy-pasted README

🔍 **Symptom:** CLAUDE.md is 60% CONTRIBUTING.md + 40% README copy.

💊 **Fix:** Distil rules · never prose. Or `@import` the source of truth.

⏱ **Cost:** Drift — the two files diverge in a month.

AP6

No PR review for CLAUDE.md

🔍 **Symptom:** Anyone edits, nobody reviews. Rules contradict each other.

💊 **Fix:** Treat CLAUDE.md as code. Same PR review as `src/`. Add a CODEOWNER for it.

⏱ **Cost:** Rule drift — team stops trusting Claude's output.

Sources: uxplanet.org CLAUDE.md best practices · reddit r/Anthropic best practices 2026 · medium.com luongnv89 CLAUDE.md DNA · anthropic PDF "How Anthropic teams use Claude Code".

SHIP THIS QUARTER

Twelve items. Each is a git-committed file, not a slide.

FOUNDATION · 4

- ☐ 01 **Project CLAUDE.md** — 8 sections, 100-200 lines, committed to git.
- ☐ 02 **Personal ~/.claude/CLAUDE.md** — 30-80 lines of your defaults.
- ☐ 03 **.claude/settings.json** — team permissions, MCP servers, model defaults.
- ☐ 04 **.claude/settings.local.json in .gitignore** — secrets never leak.

STRUCTURE · 4

- ☐ 05 **@import for deep dives** — architecture, style, testing rules split into files.
- ☐ 06 **3-6 slash commands** in `.claude/commands/` — */review* · */migrate* · */test*.
- ☐ 07 **2-5 sub-agents** in `.claude/agents/` — scoped tool whitelists.
- ☐ 08 **WIP block** at `.claude/wip/current.md` — updated weekly, deleted at sprint end.

AUTOMATION · 4

- ☐ 09 **SessionStart hook** — inject branch, WIP, last-session notes.
- ☐ 10 **PreCompact hook** — save decision log before compaction eats it.
- ☐ 11 **Stop hook** — write session summary to `.claude/state/log.md`.
- ☐ 12 **CODEOWNER for CLAUDE.md** — every change requires PR review.

Ship rule: if a checklist item isn't a link to a file in the repo, it doesn't exist. Slides & wikis don't count.

Sources: mcp.directory/blog/claude-code-best-practices · anthropic.com "How Anthropic teams use Claude Code" · agentfactory.panaversity.org CLAUDE.md hierarchy · morphllm.com Claude Code hooks 2026.

THE KPI NOBODY TRACKS

If you can't measure your memory, you can't trust it. Two KPIs. One weekly test.

TWO KPIS THAT MATTER

KPI #1 · Context recall rate

How often per week does Claude ask a question *whose answer is in your CLAUDE.md*?

Target: < 1 per week per engineer.
Method: log each "asked-again" event to `.claude/state/recall-misses.md` via a hook.

KPI #2 · Memory token tax

How many tokens does your loaded memory hierarchy consume on every prompt?

Target: < 5% of context window (~10k on 200k models).
Method: `/memory` command shows loaded files. Sum sizes. Or a `SessionStart` hook that reports the total.

THE DELETE-AND-TEST METHOD

Weekly exercise. Takes 5 minutes.

Step 1. Rename `CLAUDE.md` → `CLAUDE.md.bak`.

Step 2. Start a fresh session. Give Claude a real task from this sprint. Watch what it asks.

Step 3. Score:

- **Rules-shaped questions** ("how do you handle X?") = the file is doing its job.
- **Trivia questions** ("what's your DB?") = missing critical section.
- **No questions** = your file might be redundant with the codebase.

Step 4. Restore. Add the missing rule. Delete the redundant one. Rinse next week.

Method synthesised from field practice — anthropic.com "How Anthropic teams use Claude Code" · youngleaders.tech memory management · uxplanet.org CLAUDE.md best practices.

TRY IT THIS WEEK

Delete your CLAUDE.md. Reopen Claude Code. Give it your next real task.

What does it get wrong?

"NOTHING"

Your CLAUDE.md is redundant with the codebase. **Next play:** cut it in half. Everything that survives, earns its tokens.

"A FEW THINGS"

Healthy. Add those rules, delete the ones Claude never used. **Next play:** run this test *every Friday*.

"EVERYTHING"

You have no working memory. **Next play:** start with the 8-section anatomy on slide 06. Aim for a 120-line survivor in 30 days.

The rule: memory is a *system*, not a hope. Files curated. Hooks wired. Weekly test. That's the difference between "Claude forgot" and "Claude picks up where I left off".