From Chatbots to Coding Agents

Context, tokens, and working well with AI coding tools

SEAN DAVIS · 2026-06-18

You already know the hard part

You use chatbots. You write prompts. You judge whether the answer is any good.

That intuition transfers directly. This session adds three things:

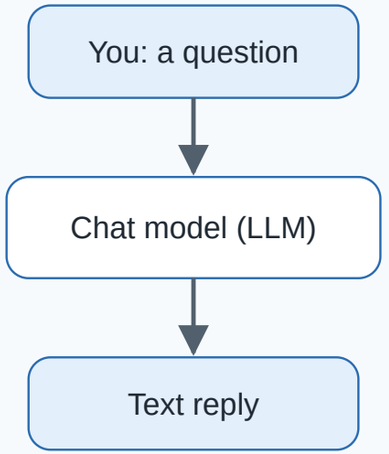
- what changes when the AI can **act in your files**, not just chat
- what is actually happening under the hood — **context** and **tokens**
- a few habits that make these tools **cheaper and sharper**

Part 1 — Chatbot vs. coding agent

Two ways to use the same model

Chatbot

answers from a prompt window

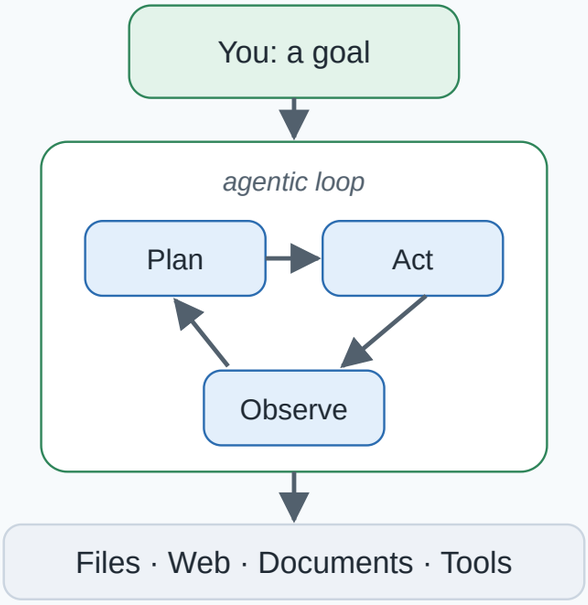


✗ no access to your files

copy · paste · adjust · repeat

AI agent

acts in your environment



✓ reads & changes your real files

Why this matters for research work

So much of our day is **file-shaped**:

- scripts, notebooks, Quarto / R Markdown
- CSV / TSV data, config and environment files
- package metadata, Snakemake / Nextflow pipelines, test fixtures

A chatbot reasons about a *snippet you paste*. An agent reasons about the *files as they actually are*.

> “Open this repo, find the script that reads the TSV, explain how the columns are normalized, then add a validation check before write-out.”

Not just code — knowledge work too

Deep research

Read many sources and synthesize a cited summary.

e.g. compare 10 methods in a table

PDF extraction

Pull structured data out of stacks of PDFs.

e.g. 40 applicant CVs → a ranked CSV

Slides & decks

Draft and edit presentations from your notes.

e.g. these very slides

Document editing

Revise grants, manuscripts, and theses.

e.g. tighten this Aims page; flag gaps

Brainstorming

Think with your papers and decks as context.

e.g. from 3 papers, propose experiments

Same idea as code: give it the **real material** — PDFs, data, drafts, slides — not a description of it.

The right mental model

A **capable but junior colleague** — not an oracle.

- reads fast, drafts quickly, runs commands, follows instructions
- also: makes mistakes, needs clear direction, benefits from review

Your job shifts:

- from writing every line → **directing what to write and why**
- from remembering context → **providing context clearly**
- from fixing alone → **reviewing output like a pull request**

Model vs. agent (one quick distinction)

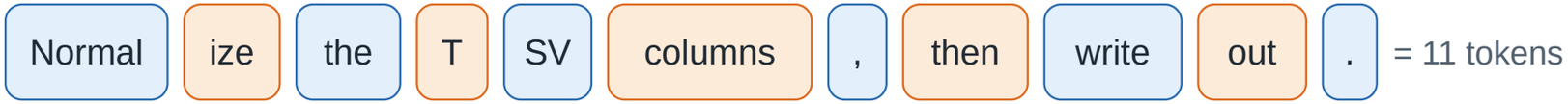
- The **model** (LLM) is the reasoning engine: predicts, plans, explains.
- The **agent / framework** is the layer that gives it tools, memory, permissions, and a control loop.

You can often swap the model and keep the workflow. *Model = intelligence. Framework = the workflow that puts it to work.*

Part 2 — What’s under the hood: context & tokens

The model reads tokens, not words

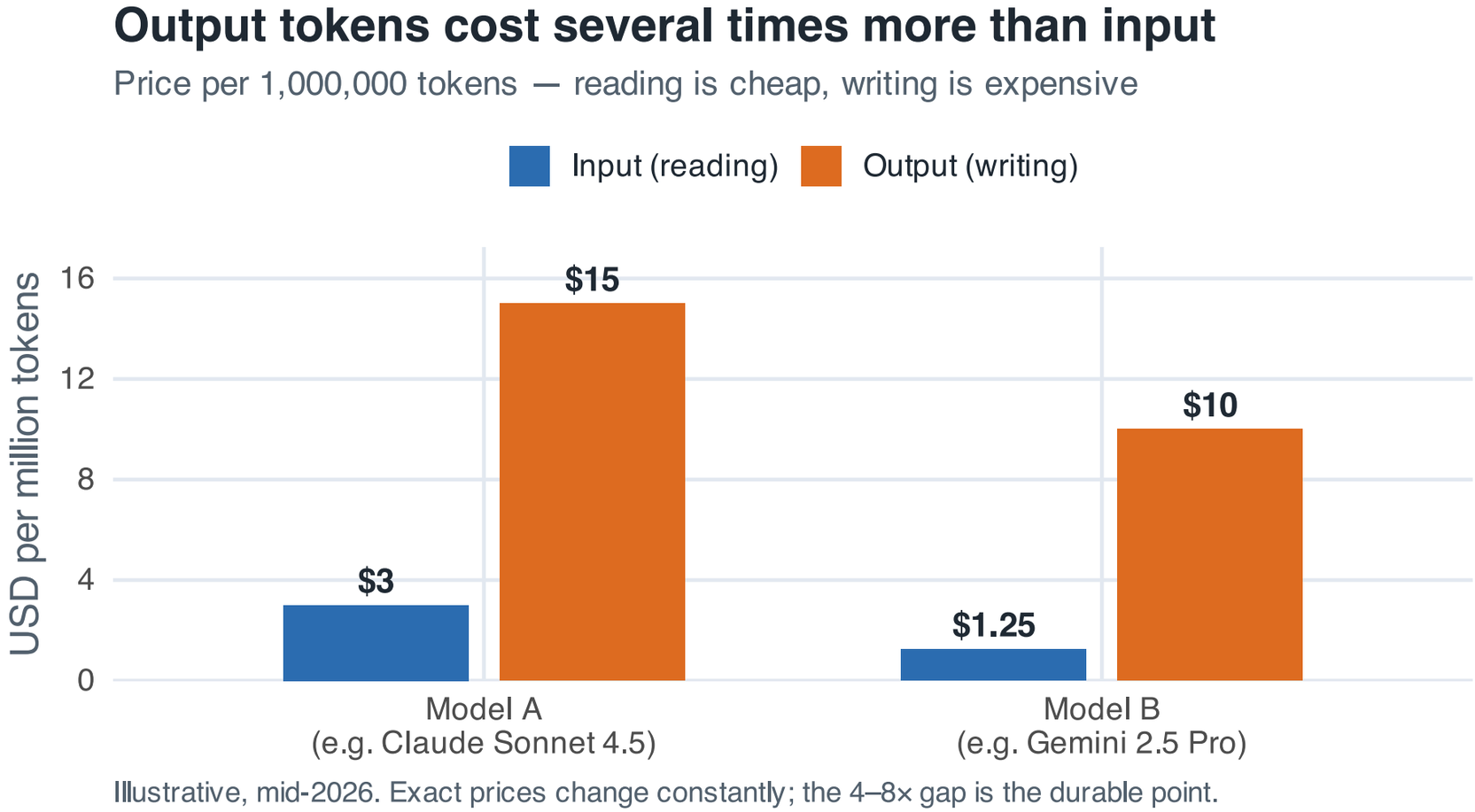
A model reads **tokens** — short chunks of text, not whole words:



1 token ≈ 4 characters ≈ ¾ of a word → 100 tokens ≈ 75 words

Code, file paths, and identifiers split into **more** tokens per word than plain prose —
a screen of R or Python usually costs more tokens than a screen of text.

Input tokens vs. output tokens



Reading is cheap; **writing is expensive**. The size of the *answer* matters, not just the size of the *prompt*.

The context window is a budget

The context window: a fixed token budget

Prompt, files, history, and the answer all compete for the same space.

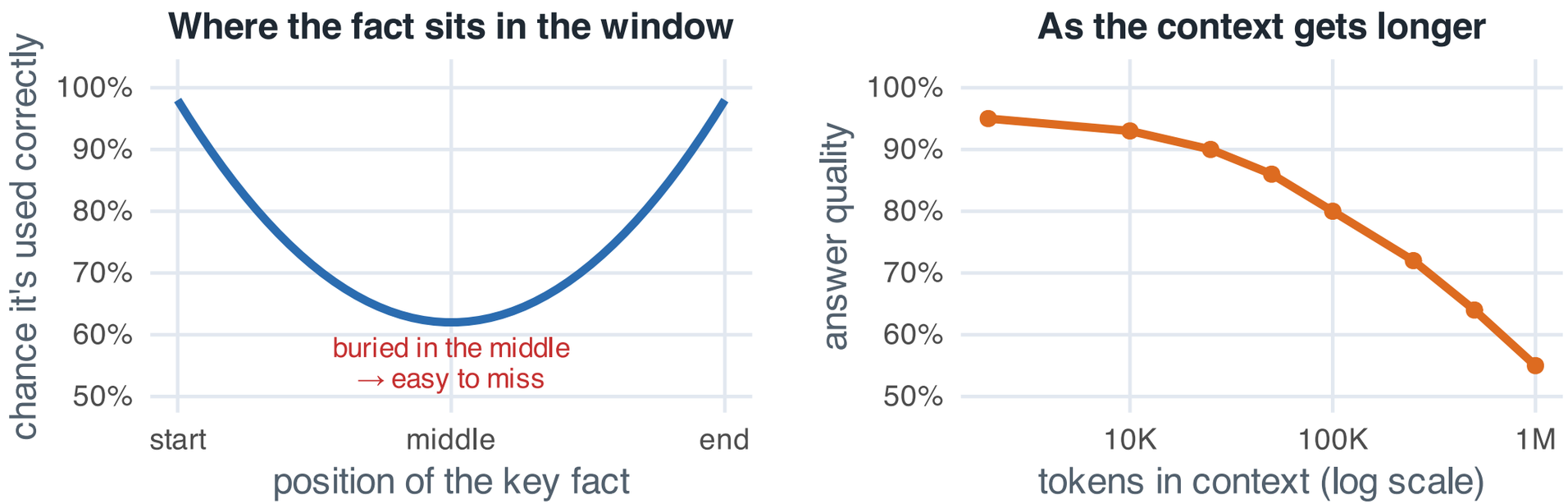
total ≤ the window limit (a few hundred K up to ~1M tokens, mid-2026)

Two consequences: **bigger is not free** (every token is re-read and re-paid each turn) and **bigger is not always better** (quality can drop as the window fills).

More context is not always better

More context is not always better

Schematic of two robust findings: "lost in the middle" and degradation with length



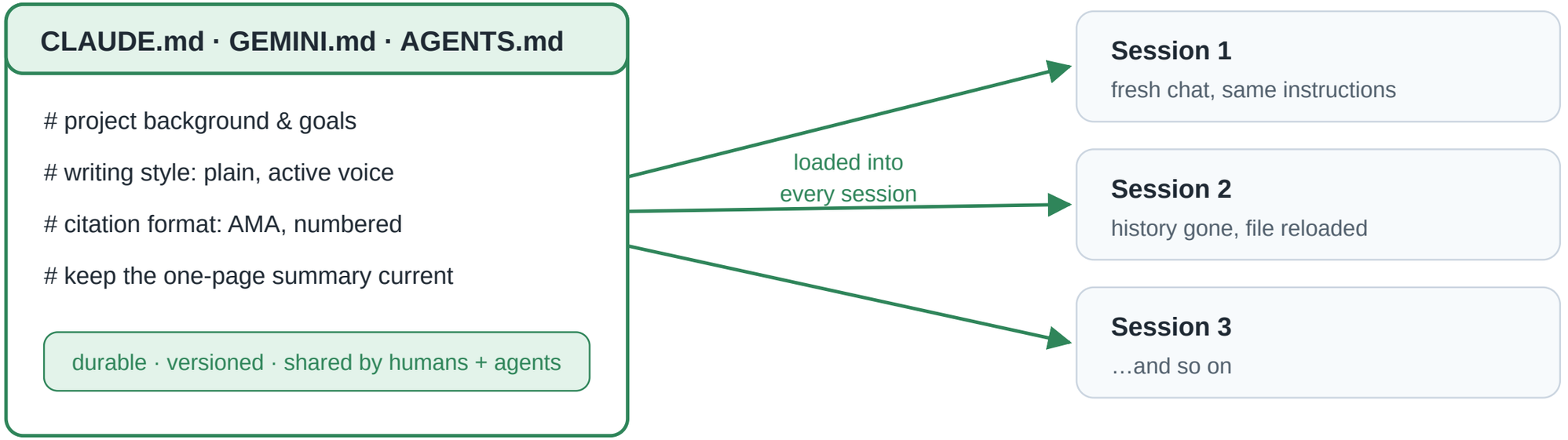
Schematic, based on Liu et al. 2023 ("Lost in the Middle") and 2025 "context rot" studies. Curves are illustrative.

Part 3 — Working well with context

Markdown as memory

Markdown as memory

Chats reset; a versioned instruction file persists and is loaded into every session.



stable file → **cached**: re-read at ≈ 10% of normal cost

Manage context: clear between ideas

One long session — context keeps growing

slower,
pricier,
confused

Clear or compact between tasks — context stays lean

focused,
cheaper,
sharper

Rule of thumb: **one objective per conversation.** Capture anything durable in a Markdown file first.

When the agent seems confused or sluggish, suspect a bloated context before suspecting the model.

Skills vs. one giant instructions file

One large instruction file

everything loaded into context, every task

- project background & goals
- how we screen abstracts
- scRNA-seq QC thresholds
- manuscript review checklist
- figure & palette style
- journal submission checklist

heavy: all of it in the window even to fix one sentence

Small core + skills on demand

stable core always loaded; playbooks load only when relevant

core instruction file (always loaded)

goals · conventions · standing preferences

small & stable → cheap to cache

abstracts	**scRNA-seq QC**	submission
skill	loaded now ✓	skill

task: "re-run our scRNA-seq QC"

→ only the matching skill enters the context

core stays small; the other playbooks cost nothing until needed

And reaching beyond your files: MCP

MCP (Model Context Protocol) — a standard way to connect an agent to external tools and data.

- GitHub, a database, an internal docs system, a lab-specific tool
- think “USB-C port for AI applications”: one standard instead of many one-offs

Putting it together

What a good session looks like

- **One objective** per conversation; clear or compact before the next
- Open only the **files the task needs**
- Put durable facts in a **stable instruction file** (cheap to cache, easy to share)
- Put task-specific procedures in **skills / separate docs** that load on demand
- **Review the output** like a pull request — you’re the senior colleague
- Watch tokens like compute time: discovery on a cheaper model, the critical pass on a stronger one

Try it this week

Start small and concrete — not “build a web app.”

- A tiny **hello-world** edit, then ask for a test, then an R-package refactor
- **Explore a Bioconductor repo** (DESeq2, GenomicRanges, …): trace one function
- **Structure a messy text file** into clean CSV + a problems report
- **Audit a small CLI workflow** for assumptions that break on another machine
- **Survey foundation models** for single-cell / spatial transcriptomics
- **Knowledge work:** extract a folder of PDFs into a table, or draft a deck from notes

Free today — $20 to keep going

- **Start free.** Everything today works on a free tier.
- **Free tiers are demo-grade** — great for learning, but they hit a wall on sustained work.
- The on-ramp: **Claude Pro — $20/mo** ($17 annual), **Claude Code included**, cancel anytime.
- A *one-month rental* of a pro tool — cheaper than any textbook, and often **expensable** (PI / training-grant funds).
- A **fork, not a hurdle**: today needs nothing; *continuing* is worth $20.

Takeaways

> Agentic tools are most useful when the work depends on the state of **real files, real commands, and real project context.**

- They **act**, they don’t just answer
- Just as useful for **knowledge work** — research, PDFs, slides, editing
- **Tokens and the context window** explain cost *and* quality
- Keep context **lean**; keep durable knowledge in **Markdown**
- Treat the agent as a **junior colleague** you direct and review

Resources

- Handout — full notes + the starter projects
- Antigravity: https://antigravity.google · Claude Code: https://claude.com/product/claude-code
- Model Context Protocol: https://modelcontextprotocol.io/
- “Lost in the Middle” (Liu et al., 2023) · “Context Rot” (2025)

Token/price/context figures are mid-2026 and illustrative; exact numbers move fast.