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**

Speaker notes

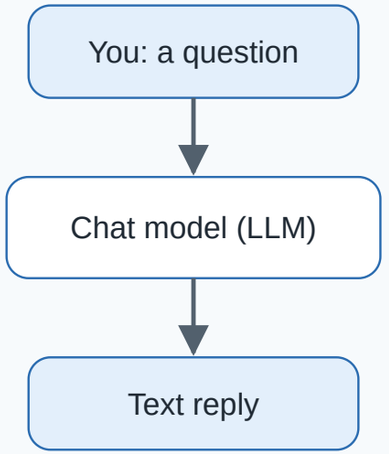
Anchor to what they already do. Most of the audience uses ChatGPT/Claude/Gemini as a chat window. The goal is not “learn to code differently” — it’s to understand a new interface and a handful of mental models. Keep it concrete and tool-neutral.

Part 1 — Chatbot vs. coding agent

Two ways to use the same model

Chatbot

answers from a prompt window

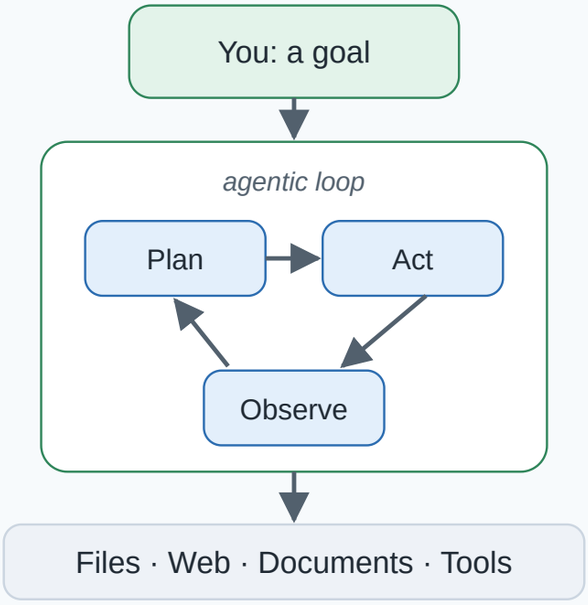


✗ no access to your files

copy · paste · adjust · repeat

AI agent

acts in your environment



✓ reads & changes your real files

Speaker notes

The model is the same kind of thing in both columns. The difference is the scaffolding around it. A chatbot answers from a prompt window — you copy, paste, adjust, repeat. A coding agent runs a loop: plan an action, call a tool (read a file, run a command), observe the result, and continue — and it does this inside your actual repository, with permissions.

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.

Speaker notes

The same capabilities — read files, follow instructions, call tools, hold context — power document work too: deep research with citations, PDF extraction into tables, drafting a deck from notes, tightening a grant Aims page. The thread is the same: give it the real material, 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**

Speaker notes

This framing calibrates frustration. When the agent makes a wrong assumption, the fix is usually a clearer prompt or better context — not a different tool. The bottleneck is almost always the quality of the instructions.

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.*

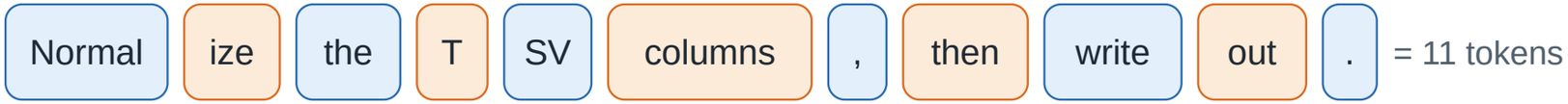
Speaker notes

Tools you'll hear about: Claude Code, Codex, Google Antigravity (which replaced Gemini CLI) and Aider as finished agents; LangGraph, PydanticAI, smolagents as frameworks people build on. The point is just that "which model" and "which agent" are separate questions.

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:



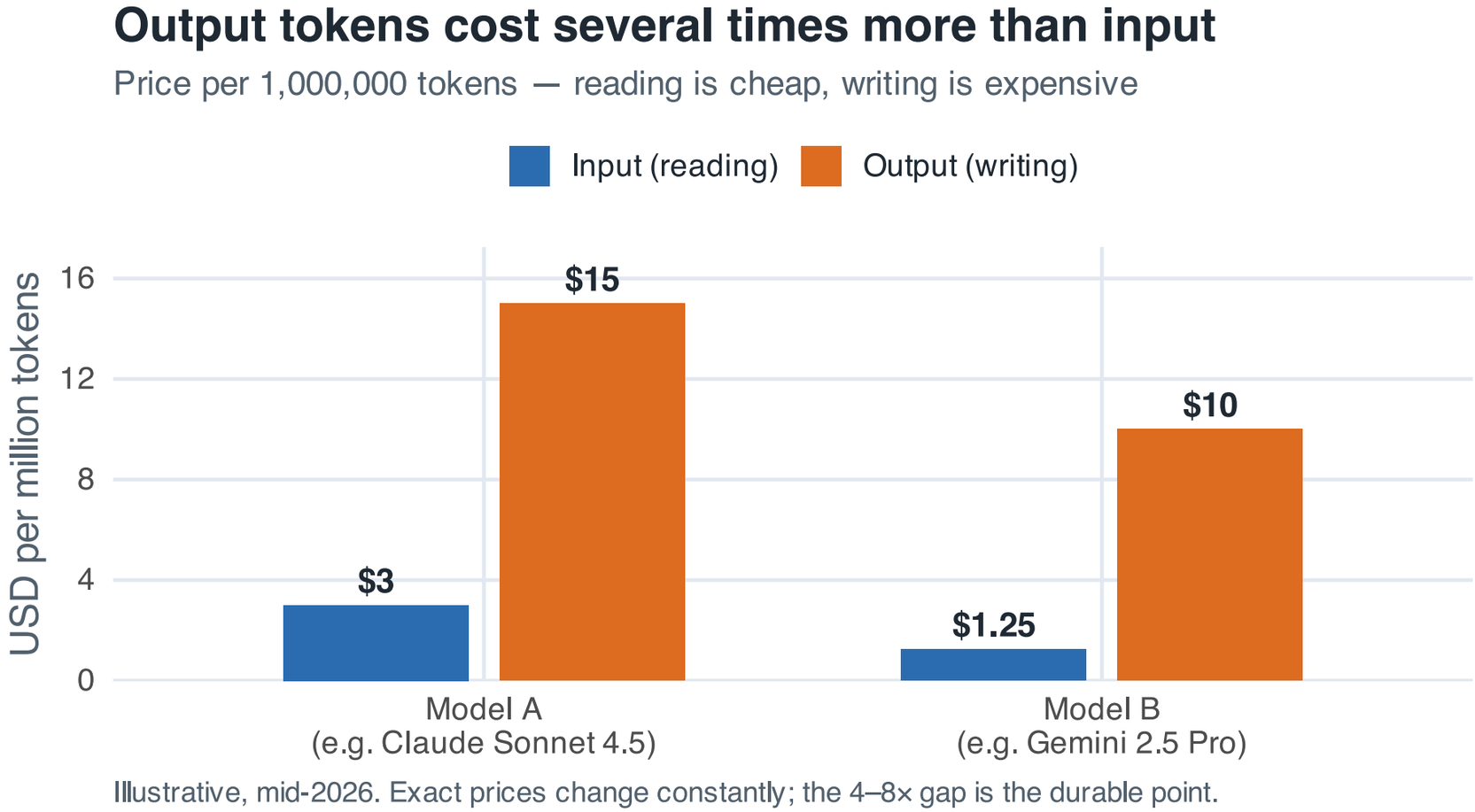
1 token ≈ 4 characters ≈ ¾ of a word → 100 tokens ≈ 75 words

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.

Speaker notes

Tokens are the unit the model actually processes. The ~4-chars / ¾-word rule is good enough for planning. The practical takeaway: code and paths are “token-expensive” — pasting a whole notebook costs more than it looks.

Input tokens vs. output tokens



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

Speaker notes

This is the cost intuition most people miss. Output is typically 4–8× the input price. So a focused, well-scoped edit usually costs far less than a sprawling rewrite, even when the model reads the same files. Numbers are mid-2026 and illustrative — the shape is the durable point.

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).

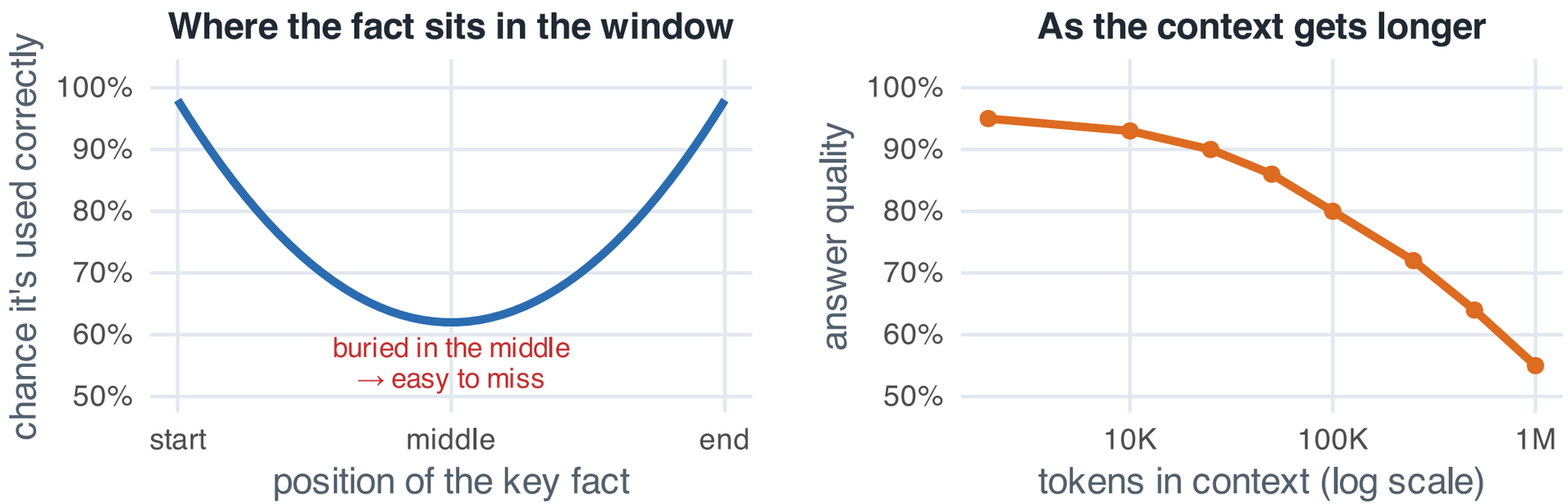
Speaker notes

Everything competes for the same space: instructions, files, the running conversation, and the answer being generated. Two consequences drive every habit that follows: bigger is not free (you re-read and re-pay the whole window each turn), and bigger is not always better.

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.

Speaker notes

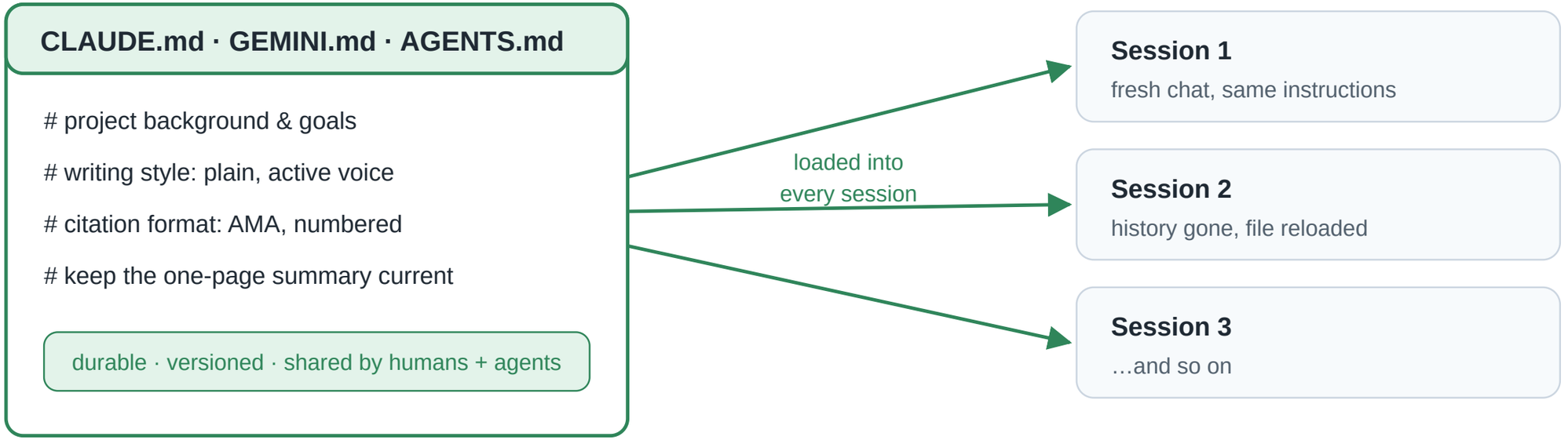
Two robust findings. “Lost in the middle” (Liu et al., 2023): models attend best to the start and end of the window and can miss material buried in the middle. “Context rot” (2025 studies across many frontier models): answer quality can degrade as input grows long — sometimes well below the advertised limit. A 1M-token window does not guarantee 1M tokens of reliable attention.

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

Speaker notes

The single most useful habit. Chats reset; a versioned Markdown file persists. `CLAUDE.md`, `GEMINI.md`, `AGENTS.md` are the same idea under different names: repository-scoped instructions that both humans and agents read. Bonus: because the file is stable, prompt caching makes re-reading it cheap. If a decision matters repeatedly, put it in a file instead of re-typing it into chat forever.

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.

Speaker notes

A long session accumulates stale files, dead tangents, and old tool output — all re-read every turn and all competing for attention. The fix: one objective per conversation; `/clear` or `/compact` between unrelated tasks; capture durable facts in Markdown 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

Speaker notes

As you encode more knowledge, you face a choice. A single big file is easy to reason about, but everything in it sits in the window on every task. “Skills” (and similar modular conventions) split knowledge into named, task-specific bundles that load only when relevant. Rule of thumb: broadly-true facts go in the small, stable core; task-specific procedures become skills that load on demand.

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

Speaker notes

Keep this light. The point is just that agents aren’t limited to the local filesystem — there’s a growing standard (MCP) for plugging in external capabilities.

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

Speaker notes

This slide is the synthesis — it ties the “under the hood” and “working well” sections into a practical checklist. If they remember one operational thing, it’s “one objective per conversation, durable facts in a file.”

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

Speaker notes

These projects are in the handout (README) with copy-paste prompts. Point them to the handout rather than reading the prompts aloud.

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.

Speaker notes

Be honest and non-coercive. The equity risk is not the $20 tool — it’s the free path quietly failing, so “optional” becomes “you had to pay to keep up.” Scope today to fit inside the free budget, and frame the subscription as “to keep doing real work after this week,” not “to survive today.”

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

Speaker notes

Land the plane on the closing idea. Everything else is detail in service of this one sentence.

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.