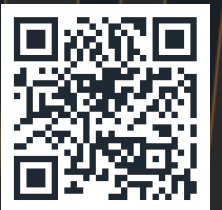


AI Agents for Epigenetics Research



What changes when the AI can act on your files, your data, and your analyses

SEAN DAVIS, MD, PHD · 2026-09-24



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.

How today works

A talk this morning. This afternoon, a book you work through at your own pace.

10:00	The talk — from chatbot to agent, under the hood, working well, reaching outside your files
12:00	Lunch
13:00	The book — hands-on, your own material, with me in the room
14:30	The Vahedi lab's afternoon

Scan this now. The book is where this afternoon happens: setup, seven chapters, every prompt with a copy button. Do its **Setup** chapter over lunch if you can.



Speaker notes

Two things to set up before starting.

The shape of the day. This morning is a talk, and nobody needs a laptop open: there are ideas, a demo or two, and three short conversations with a neighbour. The hands-on part is the book at 13:00, and each discussion slide says which chapter picks it up. Say that out loud, or half the room will spend the morning trying to follow along on a laptop.

The handover at half past two. The afternoon belongs to the Vahedi lab, uses their datasets, and is optional in a way the morning isn't. Nobody should be sitting there at 14:25 wondering whether they're allowed to leave.

The QR code goes to the book, not this deck, because the book is what people need this afternoon. The deck is on the talks site; the small code on the title slide points there.

On Setup: it takes twenty to twenty-five minutes, so lunch is the obvious time. It ends with a readiness check. The agent looks at what's installed, installs nothing, and writes `readiness.md` with a verdict for each chapter that needs a tool. If someone's setup fails, or their laptop won't let them install anything, they should pair with a neighbour at 13:00 rather than debug alone. Two people at one laptop works fine for the book.

PART 1

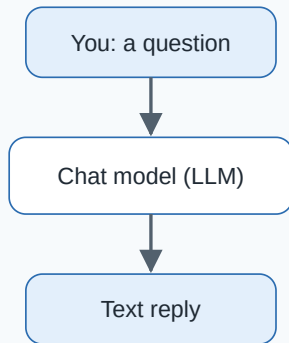
From chatbot to agent



Two ways to use the same model

Chatbot

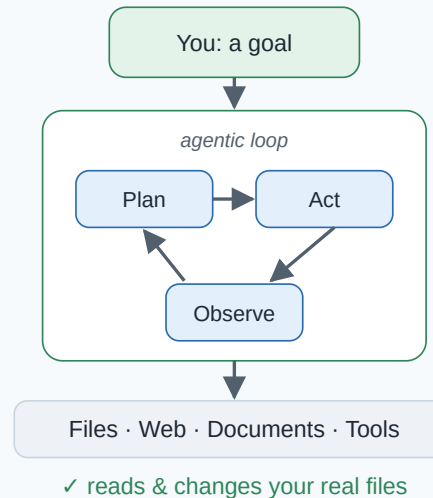
answers from a prompt window



✗ no access to your files
copy · paste · adjust · repeat

AI agent

acts in your environment



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.

Your work is file-shaped

So much of the work in this institute already lives in files:

- peak calls, count matrices, DESeq2 tables, bigWigs and their indexes
- sample sheets, submission manifests, GEO and ENCODE metadata
- pipeline configs — Snakemake, Nextflow, `params.yaml`
- manuscripts, figure scripts, methods sections, reviewer responses

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

“Take this peak table, tell me what’s in it, flag anything that looks wrong, and make me a volcano plot.”

Speaker notes

What changes here is the unit of work, not how clever the model is. A chatbot answers questions about text you've already pasted in, which means you've already done the tedious part: deciding what mattered, pulling it out, and cutting it down to fit in a chat box. A lot of the drudgery in this kind of work is that step.

The examples are meant to be dull and familiar. Every one is a file type someone here opened this week. Don't swap in something flashier; the point lands better when the material is boring and theirs.

Read the quoted prompt aloud. It's a goal in one sentence, with nothing about how to get there. That's the register for the whole day, and it's how the book's first chapter works this afternoon.

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)

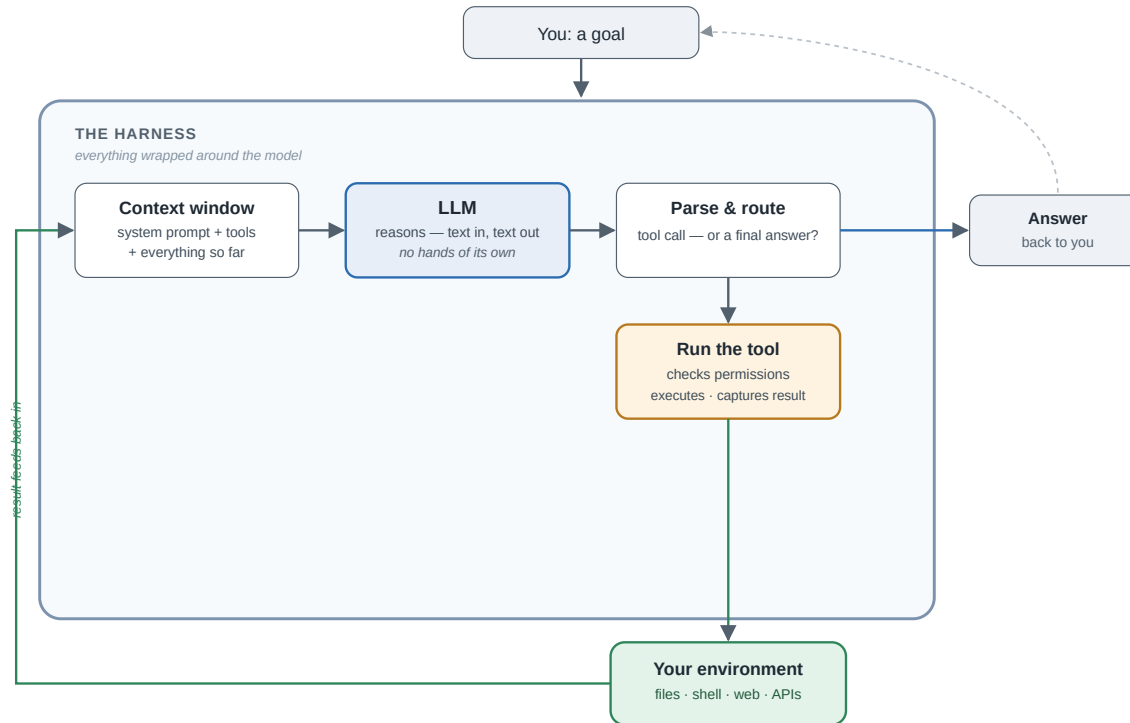
- The **model** (LLM) is the reasoning engine: predicts, plans, explains.
- The **agent / framework** is the layer that gives it tools, memory, permissions, and a control loop.

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.

What's inside the box?



The harness **assembles context**, **runs tools**, and **decides when to stop** — the model only ever sees text.

Speaker notes

This is the “agent / framework” box from the last slide, opened up. The model never touches a file, a shell, or the web — it only ever reads and writes text. Everything else on this diagram is the harness’s job: packing the context window, parsing the model’s output into an action, checking whether that action is allowed, running it, and deciding whether to loop again or hand back a final answer.

That last part matters more than it looks. “When does it stop?” is a harness decision, not a model one — some harnesses ask permission before every write, some run for many steps unattended. That is the dial you are actually turning when you pick a tool, a permission mode, or a `--yo lo` flag.

The dashed round-trip on the right is the whole point: from where you sit, you asked a question and got an answer. Everything inside the box — often several loops through “run the tool” — is invisible unless you go looking for it.

The one rule for today

Whenever it gives you a result, ask:

“How do I know this is right? Show me the evidence.”

Evidence is something you can look at: the command it ran, a count, a plot, a diff. “*I checked*” is not evidence.

Fluent and correct are different properties, and nothing in the output distinguishes them for you.

Speaker notes

This slide is the spine of the day.

The important word is evidence. An agent will happily tell you it verified its work, and the claim reads exactly like the real thing. So don't ask whether it checked. Ask to see what it checked with: the command and its output, the row count, the plot, the diff. Those are things you can look at in a few seconds, and they are either there or they aren't.

The small print is the reason. A wrong answer from an agent doesn't look wrong. It's as fluent, as well formatted and as confident as a right one. Nothing on the surface tells them apart, so the only way through is evidence.

Goal, evidence, ledger

YOU

The goal

say what *done* looks like, as something you can check

THE AGENT

The work

checks itself against that endpoint, and **shows the evidence**

BOTH

The ledger

records **who did what**, and what the evidence was

The best goals end in something **checkable**. The best agent work arrives with its evidence attached. The two go together: you can't show that a vague goal was met.

Speaker notes

This is the model the rest of the day hangs on. Three jobs, and it's worth being clear about who has each one.

The goal is yours. Not the steps, which the agent is better at choosing, but what done looks like, stated so that someone could check it. "Tidy this spreadsheet" isn't checkable. "Tidy it, and list every change you made" is.

The work is the agent's, and so is its first round of checking. A good agent run doesn't end with "done". It ends with the evidence that the endpoint was met: the counts, the command output, the plot. When it can't show that, you've learned something too.

The ledger belongs to both, and it's the next slide.

The last line is the one to say out loud. Checkable goals and visible evidence aren't two separate habits. The evidence only means something because the goal said what to measure.

The ledger: who did what

The agent is a collaborator that **can't speak for itself**. When the session ends, the only account of what it did is the one somebody wrote down.

- a plain Markdown file, **one entry per piece of work that matters**, kept as you go, like a lab notebook
- it's what an honest **AI-use statement** is made of: **who** did what (the agent, you, or both), and on what evidence
- the agent drafts it; **you sign off**, because an agent's account of its own work is exactly where overclaiming happens

*The fields are a starting point, not a form: **Goal · Who did what · Evidence · Confidently wrong · Keep**. Change them until they fit how you work.*

Speaker notes

Why keep one at all? Because the agent did real work, and it can't tell anyone about it. Next week there's no session to ask. A trainee who leaves takes their memory of which parts they did themselves. The ledger is the record of how responsibility was shared: what the agent did, what the person did, what they did together, and what showed it was right.

That has a practical use now. Journals, funders and IRBs increasingly want an AI-use statement, and "I used an agent to help" doesn't say anything. A ledger entry does.

Two things about who writes it, because both surprise people. The agent drafts the entries, either when asked or under a standing instruction, because it was there for every step. The person's job is smaller and matters more: read the entry and sign off on it, especially on who did what and on the evidence. An agent describing its own work will list the checks it meant to do as checks it did. The overclaiming numbers are coming up in Part 2.

Say clearly that the fields are suggestions. The five on the slide are the book's starting point, and people should change them. Some labs will want a field for data provenance, some won't want *Confidently wrong*. What shouldn't change is the purpose: who did what, and on what evidence. And "nothing caught" is a fine entry, or people will invent failures to fill the box.

It's not a log of every prompt. One entry per piece of work someone might need to know about later. Chapter 2 of the book is the ledger, and it carries on through every chapter after.

Discussion — what would you hand it?

Three minutes, with your neighbour. Pick one thing you'd actually like done. Say the goal in one sentence — then say how you'd know the result was right.



A website	a personal or lab site — from your CV, a paper, or your ORCID
A literature review	“...and note where the sources disagree”
Your CV	what’s missing, what’s stale — then fold in the last two years
Slides from a paper	yours or open-access, with speaker notes
A messy spreadsheet	tidy it, <i>and</i> list every change you made
A data file	what’s in it, what’s suspicious, plot it
An app	a filterable browser over ENCODE or 4D Nucleome metadata

This afternoon: **Chapter 1** builds the website, **Chapter 2** its ledger, **Chapter 6** the app.

Speaker notes

The hands-on version is the book at 13:00. Here the aim is to get people naming a real task, and then answering the second question.

People work much harder on something they actually wanted done, so the menu matters more than any item on it. Tell them none of these need code. Otherwise half the room assumes they do and picks something smaller.

Take three answers. Listen for two things. One is a goal that's really a list of steps: turn it back into an outcome, because the agent should choose the route. The other is the second half, "how would you know it was right?", which most people skip. That question is the checkable endpoint, the thing the agent's evidence will be measured against, so get someone to answer it out loud.

Stop at three minutes, while people are still curious. They get the real thing this afternoon.

PART 2

What's under the hood



The model reads tokens, not words

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

Normal ize the T SV columns , then write out . = 11 tokens

1 token $\approx$ 4 characters $\approx$ $\frac{3}{4}$ of a word $\rightarrow$ 100 tokens $\approx$ 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.

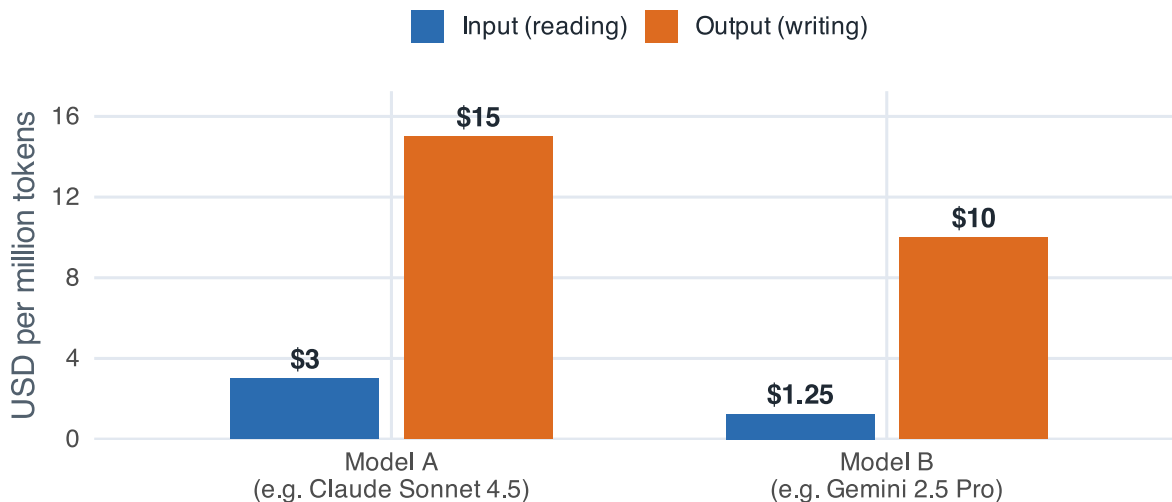
Speaker notes

Tokens are the unit the model actually processes. The ~ 4 -chars / $\frac{3}{4}$ -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

Output tokens cost several times more than input

Price per 1,000,000 tokens — reading is cheap, writing is expensive



Illustrative, mid-2026. Exact prices change constantly; the 4–8x gap is the durable point.

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 $\leq$ 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).

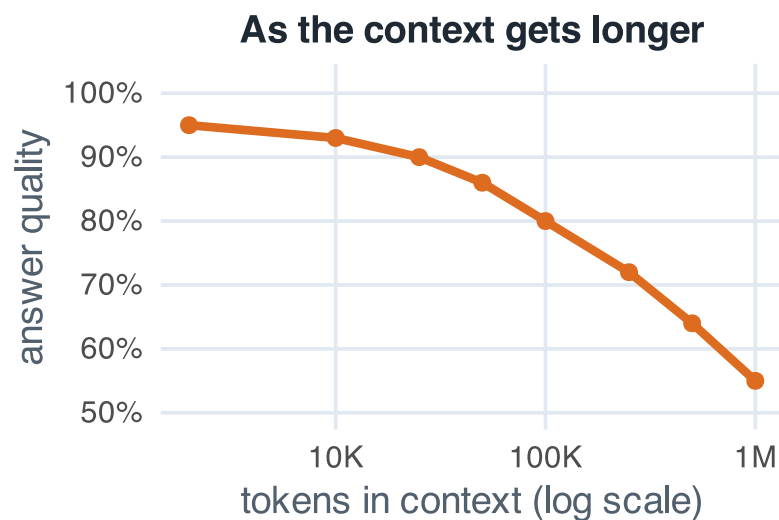
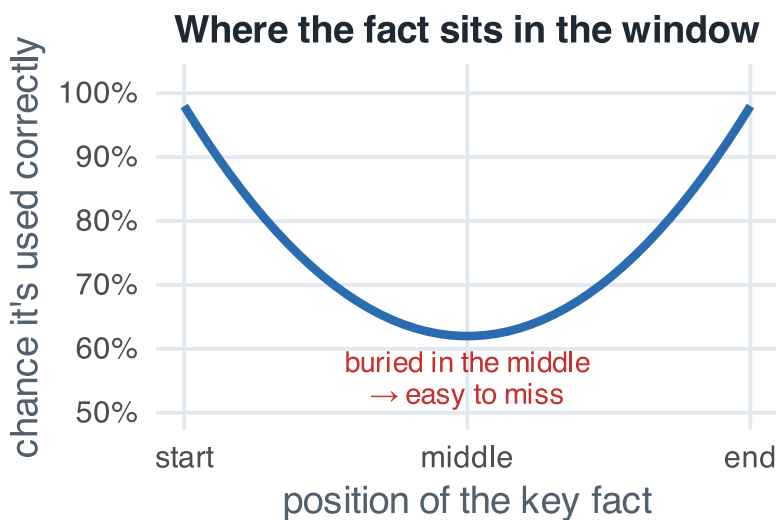
Speaker notes

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.

The model is only half of it

The same model, in two different harnesses, is not the same tool.

The model

reasons, plans, writes

picked by the vendor

changes every few months

The harness

reads files, runs commands, retries

how much it can see and do before asking you

decides whether it verifies its own work

So “*which model should I use?*” is rarely the question that matters. For the work in this room, the gap between the best model and the fifth best is **smaller than the gap between a clear request and a vague one.**

Start with whatever your institution already provides.

Speaker notes

People want a ranking, and a ranking isn't where their leverage is. A strong model in a harness that can't read your files is a chat window. A slightly weaker model in a harness that reads, runs, checks and retries will beat it on everything in this deck.

This also explains something people find confusing: the same model behaves differently in different products. That's the harness: which tools it's allowed, how it manages the context window, whether it checks before it reports.

If the question turns to benchmarks, and it usually does, there are three reasons to take them with some salt.

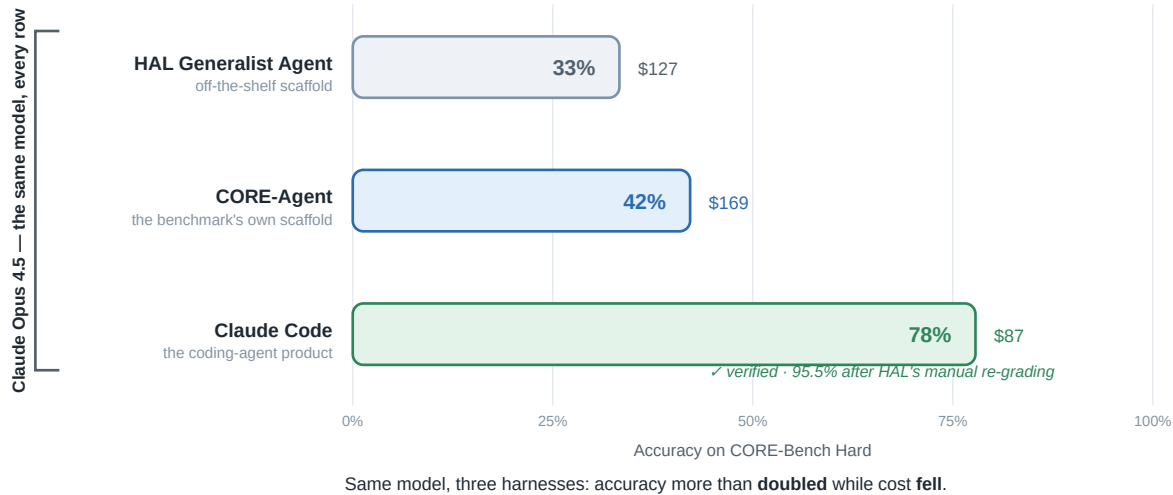
Most published benchmarks measure the model, not the harness, which is the half that matters less for this work. The agentic ones that test the whole loop are newer and noisier.

On the benchmarks people quote, the top models sit within a few points of each other, which is inside the noise of how the harness was set up. A two-point lead isn't a reason to switch tools.

And a benchmark measures what it measures. "Resolves GitHub issues in Python repositories" is a real skill, but it isn't extracting cohort definitions from forty PDFs. Check what a benchmark actually asked the model to do before you let it decide anything.

The better move is to run your own small comparison on a task you care about. Ten minutes of that tells you more than a leaderboard. The next few slides show what two evaluations found anyway. (How to read a cost–accuracy plot is in the appendix, if someone asks about leaderboards.)

Same model, three harnesses



Same weights, same weekend of benchmark tasks — the only thing that changed row to row was **what surrounded the model**.

HAL (Princeton) · **CORE-Bench Hard**: reproduce a published paper's results from its code and data · three scaffolds, one model, read straight off HAL's own leaderboard rows¹

Speaker notes

This is the harness slide, measured.

HAL, the Holistic Agent Leaderboard from Princeton, scores every run for accuracy and for what it cost. CORE-Bench Hard is the benchmark closest to this room: each task is a published paper, and the agent has to reproduce its results from the paper's code and data, including getting the dependencies to install.

All three bars are the same model, Claude Opus 4.5. HAL's generic scaffold solved 33% for \$127. CORE-Agent, the benchmark's own scaffold, got 42% for \$169. Claude Code got 78% for \$87, the best result and the cheapest. HAL notes that after they hand-checked a few grading errors, the Claude Code run reached 95.5%, and they declared the benchmark solved.

Two caveats. These are 2025 models, and HAL has paused adding new ones, so treat this as a snapshot, not a shopping list. And the Claude Code run was submitted by someone else, not run by HAL. Neither changes the point: the same model, with better tools and a better loop, did twice as well for half the price.

What K-Bench is actually testing

- Built from **178 real first messages** sent to K-Dense's scientific-agent platform — underspecified, often with attachments, **no answer key**
- Each ran through **nine frontier models**, same sandbox, same request → **1,602 runs**
- **Three separate model judges**, blinded to which model produced the run, score every one on eight dimensions — scientific accuracy, communication, data handling, and more

Not “*did it match the reference answer*” — “**would a scientist accept this?**” Eight out of ten means yes, with minor edits.

K-Bench 01 (K-Dense)²

Speaker notes

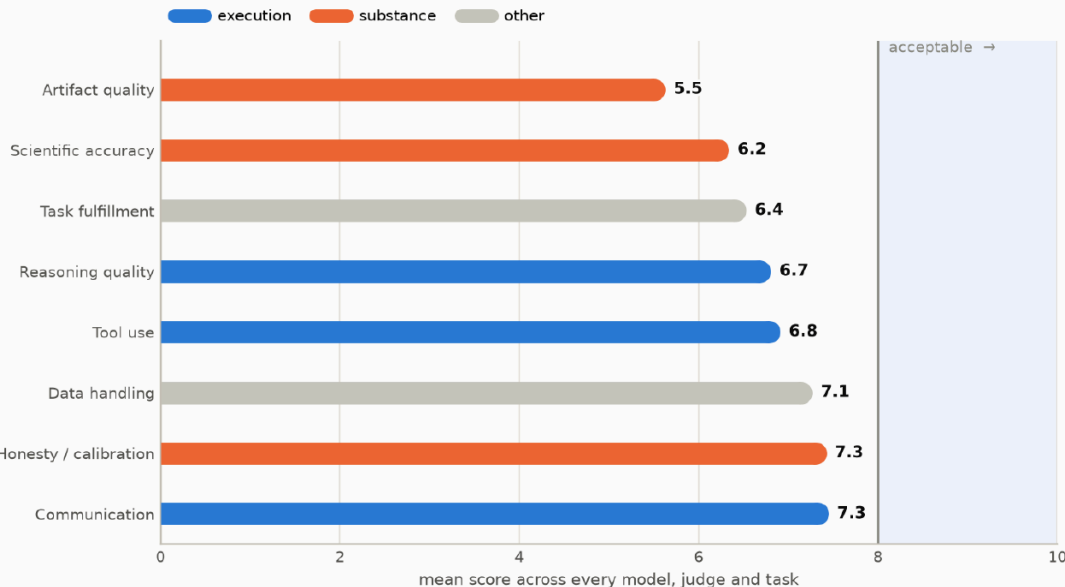
HAL and CORE-Bench grade agents against a known target: a paper's numbers, reproduced or not. K-Bench has no target. Real scientists sent real, messy requests to a working research platform, and nobody knew in advance what a correct answer looked like. That's what "no answer key" means.

So the authors judged quality instead, on eight dimensions a working scientist would care about, with the judges blind to which model produced each run. The next two slides are what they found.

Competent operators, weaker scientists

Competent operators, weaker scientists

Execution dimensions — driving the tools, structuring the answer, reasoning through it — average 6.9. The substance dimensions the science actually rests on average 6.3. Nothing reaches the acceptable line.



Scientific accuracy 6.2, communication 7.3 — in *every one* of the nine models. **Nothing** reaches “acceptable” on average.

K-Bench 01 · average score by dimension, sorted · nine models, 1,602 runs²

Speaker notes

Eight on the K-Bench scale means “a domain scientist would accept this with minor edits”.

The chart averages across all 1,602 runs, sorted. The dimensions that make an answer look good score highest: communication, honest tone, data handling. The ones the science rests on score lowest: scientific accuracy, and the quality of the files it produced. The slide title is the authors’ own title for this figure. And the gap between accuracy and communication runs the same way in all nine models.

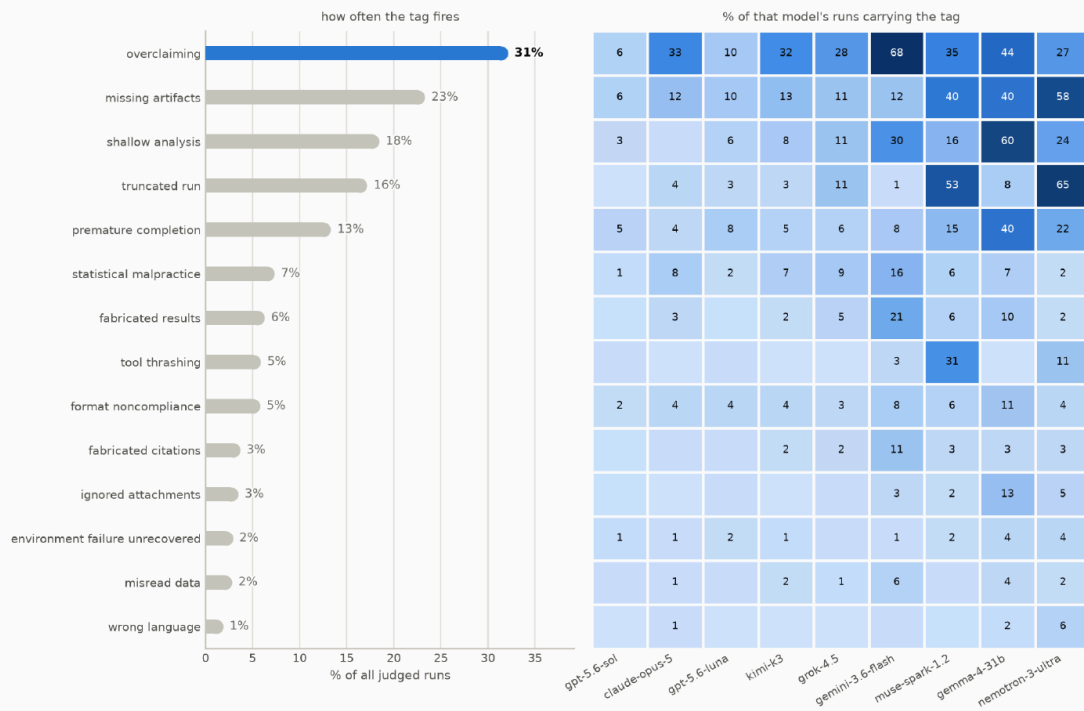
The caveats go in the same breath. The judges are models, not scientists. The authors say the reproducible result is the ordering of the models, and that the absolute scores reflect their rubric. They also run the platform the requests came from.

One finding cuts against something I said a few slides ago. On any single request, which model you used mattered a lot: across models, scores on the same task differed by more than six points on average. So on the task in front of you, trying a second model is often worth it.

The most common failure is overclaiming

The most common failure is misrepresentation, not incompetence

Judges tagged each run from a closed 16-tag taxonomy; tags are not exclusive, so rows do not sum to 100%.
The leading tag is 'overclaiming' at 31% of runs. Tags under 2% everywhere are omitted.



31% of runs overclaimed — more than missing files, shallow analysis or fabricated results. Back to the one rule: *“How do I know this is right? Show me the evidence.”*

K-Bench 01, failure tags per run · left: all runs · right: by model²

Speaker notes

The judges tagged each run from a fixed list of sixteen failure types. The most common, across everything, is overclaiming: saying more was done, found or checked than the run supports. The authors put it well: the most common failure is misrepresentation, not incompetence.

Point at the right-hand panel. It isn't only the weak models. Some of the strongest do it in a third of their runs, and the cleanest are at six to ten percent. Nobody here can assume their model is one of the clean ones on their task.

This is why the rule asks for evidence rather than an account, and why a person signs off on the ledger even though the agent drafts it. An agent describing its own work is where overclaiming happens.

PART 3

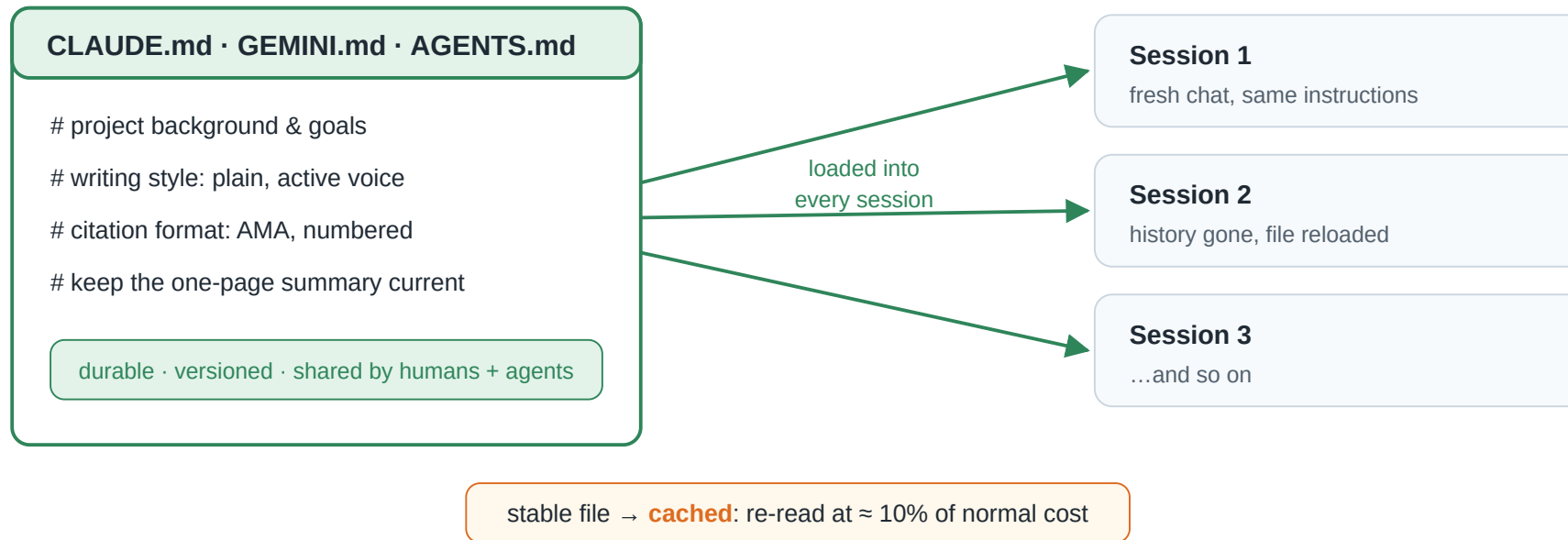
Working well with an agent



Markdown as memory

Markdown as memory

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



Speaker notes

If people take one habit from today, it's this one. A chat resets when it ends; a Markdown file in the project stays. `CLAUDE.md`, `AGENTS.md` and `GEMINI.md` are the same idea under different tools' names: instructions for the project that people and agents both read.

A side benefit is cost. Because the file doesn't change, the tool can cache it, so reading it every session is cheap. If a decision matters more than once, put it in a file instead of typing it into chat again.

Manage context: clear between ideas

One long session — context keeps growing



Clear or compact between tasks — context stays lean



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 collects stale files, abandoned tangents and old tool output. All of it is re-read on every turn, and all of it competes for the model's attention.

Keep one objective per conversation. Between unrelated tasks, start a new session (in Claude Code, `/clear` does it) after writing anything worth keeping into a file. When the agent seems confused or slow, suspect a bloated context before you suspect the model.

How epigenomics data misleads an agent

All real, and all in the files you have:

What it looks like

What it actually is

`chr` column reads `12`, `X`

Ensembl-style names — **silently joins to nothing** against UCSC

`padj` is `NA` for 20,982 genes

DESeq2 independent filtering; `padj < 0.05` **drops them all**

A peak 93,567 bp wide

Union peak set merged neighbours — not one regulatory element

`pvalue` of exactly `0`

Floating-point underflow; `-log10` gives **infinity**

`log2FoldChange` of `-1.13`

Higher in the *knockout* — and **nothing in the file says so**

Each is a **checkable endpoint** you can ask for up front: “*report how many rows joined*”, “*state the reference level*”. Then **write it down once**, in a `DATA.md`.

Speaker notes

This is the slide written for this room. Every row was checked against the files in the workshop repository, and the counts are exact.

An agent will read past all five without pausing. So will a tired postdoc; the difference is that the agent will then write a confident methods paragraph about it.

All five have the same shape. The data is correct, and the information you'd need to read it is missing. None of these is a corrupted file or a pipeline bug. Genome build, coordinate convention, reference level and filtering history are things the analyst keeps in their head, and the agent doesn't have that head.

Spend time on the fold-change row. In the RNA-seq file, *Jun* has about twice the counts in the knockout and a log2 fold change of -1.13 , so positive means higher in wild type. In the peaks file it's the other way round. Same lab, same family of pipeline, opposite conventions, and neither file says which. That isn't carelessness. It's what DESeq2 output looks like when the reference level lives in the R script and not in the exported table.

What to leave them with: whenever you give an agent a results table, give it the four facts the table leaves out. Genome build, coordinate base, reference level, and what was filtered before export. It's one sentence in a prompt, and it prevents every error on this slide.

The closing line is the first mention of a theme the day comes back to twice more: a repository can carry memory that outlives any one session. Said here it's informal — a README, a data dictionary, whatever the lab already half-does — not yet the word “skill.” That word arrives in Part 4. Don't over-explain it now; let it land quietly and pay off later.

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.”

Say what you want, not how

The more independently it works, the more the **specification** carries. A one-line request leaves dozens of decisions open, and it **guesses every one**: unguided one-shot attempts land about **a third of the time**.

The fix costs two minutes and isn't writing a spec yourself:

“Interview me until you could do this without guessing. One question at a time. Look around first — don't ask me what you can work out.”

And end the goal on something **checkable**: “...and every *DE* call states its reference level.” That endpoint is what its evidence gets measured against.

Speaker notes

This is the habit people resist most, because it feels like overhead when the tool looks ready to start right away.

Writing a plan first doesn't make the model cleverer. It removes guesses. Every decision you didn't specify gets made somehow, and the chance that all of them land where you wanted is what gives that one-in-three figure for unguided attempts.

Flipping it round is what makes this work for this audience. Nobody here wants to write a specification. Everyone can answer questions about their own work, and the questions bring up decisions they didn't know they were making. That's why the prompt has the agent interview *them*.

Two parts of the prompt do most of the work. "One question at a time": a list of eight questions gets eight rushed answers, and each answer should change what gets asked next. And "look around first": if the answer is in the files, it shouldn't cost one of your questions. Without that line, the interview turns into reading your own folder aloud.

The last line connects this to the one rule. An interview settles the decisions; a checkable endpoint settles what done means. Give examples from the room's own work: every peak set reports how many peaks survived each filter; every figure's legend matches its code; every DE table names the reference level. Then "show me the evidence" has something definite to point at.

The book's Chapter 6 opens with a prompt built on this. The agent checks what's installed, then interviews you into a short spec before it builds anything.

Discussion — what's only in your head?

Three minutes, with your neighbour. Think of a folder you work in — an analysis, a grant, a manuscript. What would a smart newcomer get wrong on day one? Name **one** thing that isn't written down anywhere.



Don't write a file describing your project. **It can read your project.** Write down the part that's only in your head — and if your list is empty, write no file.

This afternoon: **Chapter 3** turns that list into an `AGENTS.md`, and tests it in a fresh session.

Speaker notes

Chapter 3 of the book is the hands-on version.

The second half of this slide is the content, so don't rush it. The usual advice is to start by writing the agent a long file describing your project. That advice is mostly wrong, and this room will have heard it. Agents read projects well: files, READMEs, code, data. A file that repeats what's already there costs context every turn and adds nothing.

What an agent can't work out is what's only in your head: how you like things done, what "finished" means, which of four scripts is the real one, the constraint that looks like a mistake and isn't. That's what goes in the file.

Take two or three answers. The good ones are specific and a little embarrassing: "the sample sheet's `cond` column, where 1 means knockout". Accept "I couldn't think of one" as a real answer. It's true more often than the internet suggests.

PART 4

Teaching it what you know



What the model doesn't know about your lab

- It knows genomics in general. It does **not** know that your lab flips fold-change sign, or that a `padj` of `NA` means DESeq2 filtered the gene out — not that the gene wasn't significant
- That knowledge already exists — in a labmate's head, an old protocol document, a Slack thread, “ask Priya”
- Right now it gets **re-explained, slightly differently, every time** someone asks the agent for help

The fix isn't a smarter model. It's **writing the procedure down once**, so the agent loads it exactly when it's relevant — that's what a **skill** is.

Speaker notes

This picks up the last discussion, which asked what's only in someone's head. It's the same problem, now with a fix: a skill is how that knowledge gets written down once and reused, instead of re-explained every session, a little differently, by whoever is prompting that day.

Say this for the PIs, because they won't wait for the next slide to decide whether it applies to them: nobody is asking them to become programmers. A skill is a text file. If you've written a protocol, an SOP or a README, you've written most of one.

A skill is a folder

```
lab-qc-report/  
├── SKILL.md          # required: frontmatter + instructions  
├── scripts/          # optional: code the agent runs  
└── references/       # optional: docs loaded on demand
```

`SKILL.md` needs exactly two fields: `name` and `description`.

The description is the whole game. The agent never sees the body until it has already decided to load it.

Weak	<i>“Helps with QC.”</i>
-------------	-------------------------

Strong	<i>“Runs our standard QC on a differential peak table: chromosome naming, fold-change direction, p-value underflow, peak widths. Use when asked to check, QC, or sanity-check a peaks or DESeq2 file.”</i>
---------------	--

Speaker notes

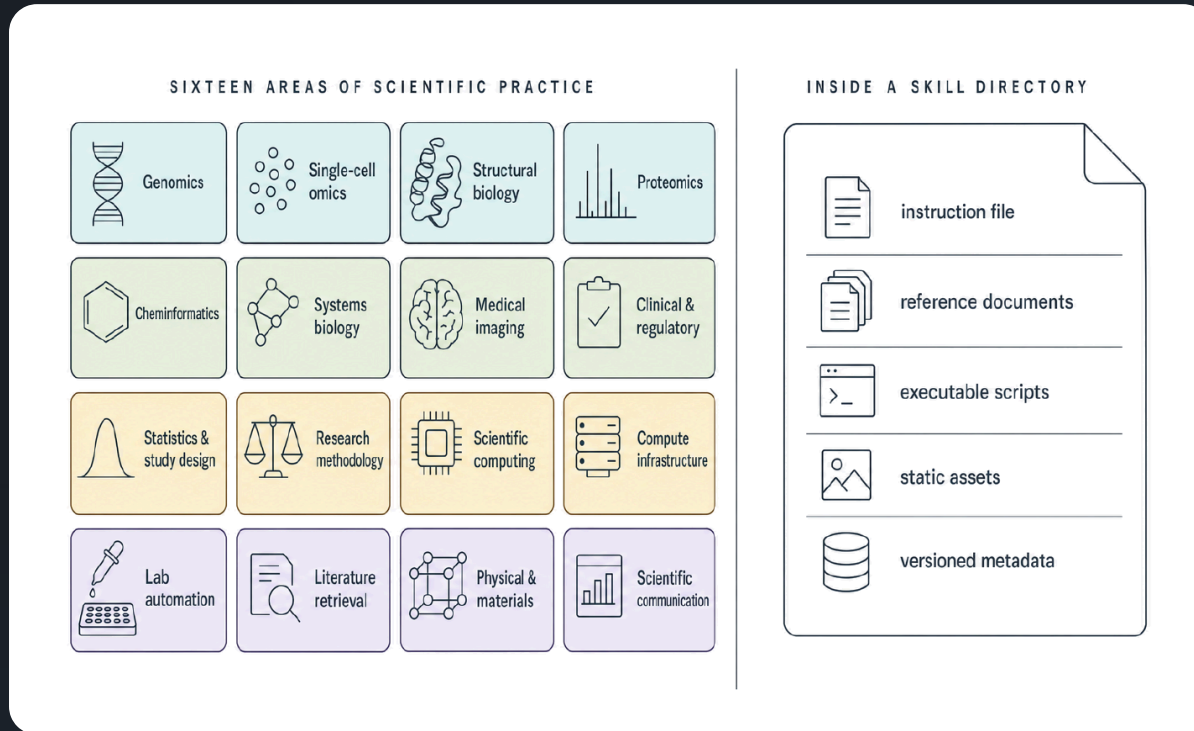
People expect a framework and find a text file. If you've written a README, you've written most of a skill. There's no build step, no registration and no code required.

Skills are cheap to keep around because of how they load. At the start of a session the agent reads only each skill's name and description, roughly a hundred tokens each. The rest of the file loads only when a task matches. So fifty skills cost about five thousand tokens of standing context, not fifty documents' worth. A context file like `AGENTS.md` is loaded for the whole session, which gives a simple rule: when a section of it has turned into a *procedure* rather than a *fact*, move it into a skill.

Warn them about the quiet failure. A skill that never loads looks exactly like a skill you never wrote. You get no error and no warning, just the answer you'd have got anyway. That's why the description matters so much, and why Chapter 4 of the book has a step for testing it.

`SKILL.md` is an open standard now, read by Codex, Google Antigravity (which replaced Gemini CLI) and Copilot as well as Claude Code. Writing a procedure this way doesn't tie you to one vendor.

A library of scientific procedures, one folder each



A good rule, written once: “3 mice with 100 cells each is $n = 3$ (mice), not $n = 300$ (cells).” And a candid paper: **“We report no task-level evaluation.”**

Scientific Agent Skills (K-Dense, MIT licence) — 166 skills today: `pydeseq2`, `bulk-rnaseq`, `scanpy`, `experimental-design` ... · paper: [arXiv:2609.00065](https://arxiv.org/abs/2609.00065)³

Speaker notes

You don't have to write every skill yourself. This is a large open library of them from K-Dense, 166 when I last looked, and the figure is from the first page of their paper: sixteen areas of practice on the left, and what's inside each skill's folder on the right. Several are close to this room's work: `pydeseq2`, `bulk-rnaseq`, `scanpy`, `deeptools` and `pathway-enrichment`.

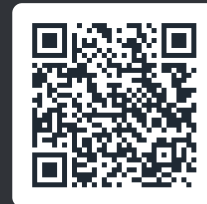
The paper opens with mistakes an agent makes when its code runs but the analysis isn't defensible: no multiple-testing correction, cells counted as replicates, a BED file's zero-based start read as one-based. For each it shows the rule in a skill that prevents it. The `n = 3` line is from their `experimental-design` skill. That's a good use of a skill: a fact the field agrees on that an agent won't reliably apply unless told.

What I like most is that the paper is honest about what it hasn't shown. The abstract says it reports no task-level evaluation, so it doesn't claim the skills make an agent's work better. Its limitations section reads like a list of this talk's warnings. Nobody has measured how often the right skill gets picked. A picked skill can still be followed badly. The descriptions of all 163 skills in the version they describe take about seven percent of a large context window before any is used. And 32 don't list the team as author. Their advice is to install only what a project needs, and to read a skill before installing it, because skills can run code.

No library can know your lab, though. `pydeseq2` knows how to run DESeq2. It can't know that two files in *your* lab use opposite fold-change conventions. Use libraries for what the field shares, and write your own skills for what only your lab knows. Chapter 4 of the book has a discussion built on this paper.

Discussion — what would you write down as a skill?

Three minutes. Which check or procedure do you do by hand, every time, that a new trainee forgets? Write it down once, as a skill — then say what file you'd test it on.



Results-table triage	encode the traps from earlier, once
Ledger summary	who did what across a project, and which evidence is only the agent's word
Reference check	does every citation resolve, <i>and</i> say what the sentence claims?
Paper → lab meeting	ten bullets for a named audience, plus the likely questions
Reviewer 2	attack the paragraph before a real reviewer does
Figure legend	bullets in, journal-style legend out

This afternoon: **Chapter 4** — the agent writes the skill with you, and you test it on a file where you already know the answer.

Speaker notes

Chapter 4 of the book is the hands-on version, with a worked example on the Vahedi lab's DESeq2 tables.

Push on the second question. Testing is what separates a working skill from a folder of hopeful Markdown, and it's the step almost everyone skips. A good answer names a file where they already know what's wrong: "last month's sample sheet, the one with the duplicated row".

Recommend the ledger summary out loud. It reads the ledger they'll build this afternoon and summarises who did what, flagging entries whose only evidence is the agent's own account, so the skill audits their own habits. It's also a good example of a skill that has to be told what *not* to do. An agent that helpfully fills in missing evidence has destroyed the one thing the ledger is for.

The same thing Anthropic ships

Claude Science is a research workbench built on exactly the mechanism on the last few slides.

- **60+ curated skills and connectors**, pre-configured for genomics, single-cell, proteomics, structural biology, cheminformatics
- wired to **UniProt, PDB, Ensembl, Reactome, ClinVar, ChEMBL, GEO**
- runs on your laptop or on the **cluster**
- and you save your own pipelines as skills, which every later session inherits

Same `SKILL.md`. Same folder. **The thing you'll write this afternoon is the unit of composition in a research platform.**

Beta, on Pro / Max / Team / Enterprise. One researcher in their write-up built about twenty custom skills for a single workflow.

Speaker notes

This comes straight after the discussion because it stops skills sounding like a toy.

The argument isn't "go and install Claude Science". It's that the file they'll write this afternoon is the same unit a purpose-built research platform is assembled from, sixty of them at a time. Skills aren't a beginner's version of something more serious. The format is the same at both ends.

Read the domains out loud, because people will hear their own field: genomics, single-cell, proteomics, structural biology, cheminformatics. The databases are ones they already use by hand, like UniProt, Ensembl, GEO and ClinVar. The HPC detail matters to the computational people: it can run on a login node over SSH, so it isn't limited to what fits on a laptop.

The caveats: it's in beta and needs a paid plan, so don't present it as something everyone can open this afternoon. And nothing on the slide depends on anyone adopting it. The claim that carries over is that the skill format is worth learning because the serious tools are built from it.

If someone asks whether their own skills work there: yes. That's what an open format is for.

Version control is the undo button

The moment an agent edits your files, `git` stops being a developer habit. It's what lets you say **no**.

- **commit before you set it loose** — then `git diff` shows exactly what the agent did
- **small, frequent commits** — each one is a point you can go back to
- **a branch costs nothing**, and makes “throw it all away” a single command

You don't type the commands: the agent already knows git. What it doesn't know is **your lab's** habits — what needs a second pair of eyes, how you name things. Same move as skills: write that down once. This afternoon: **Chapter 7**.

Speaker notes

Start with the safety argument, because it applies to everyone. Someone who has never used git, and now has a tool that rewrites their files, needs an undo button far more than a branching strategy.

A clean state before you start is what makes `git diff` an honest review afterwards. Everything it shows is what the agent did, not a mix of the agent's changes and your own unsaved edits. Small commits give you more places to go back to. A branch costs nothing, and throwing away a bad attempt becomes one command instead of an afternoon of unpicking.

Nobody has to learn git commands for this. The agent runs them; you say what you want in plain words and check what happened. The book's Chapter 7 works that way, and for anyone on a managed laptop who can't install git, it uses plain folder snapshots instead.

The small print on the slide is the same idea as the rest of this section. The agent already knows git in general. What it doesn't know is your lab's version: "never commit data files", "ask before pushing". A few lines in `AGENTS.md`, or a small skill, covers it. Most people won't think to write that down, because they assume the agent already knows, or that it isn't worth saying.

PART 5

Reaching outside your files



Its knowledge stops at training day

- Ask it about a paper from last month, a database updated yesterday, or your institution's own repository, and it either says so — or, worse, answers fluently and wrong
- On its own, an agent only has what's in the chat window and the files you've given it. It cannot reach a live database by itself

MCP (Model Context Protocol) is the plumbing that lets it call out to the actual system of record — a database, a search index, a LIMS — instead of recalling one from training.

Speaker notes

This is a different move from skills, so keep the two apart. A skill packages what's already true about your lab so the agent doesn't have to be told twice. MCP reaches for something that was never in the model at all, either because it's newer than the training data or because it's yours and was never public.

The line to get across before the demo: look it up, don't recall it. When a model is unreliable about a fact, a better prompt won't fix that. A connection to the system that already has the answer will.

Two databases, one question

“Which genes does Open Targets most strongly link to ICF syndrome, and what do recent PubMed papers say about how they converge on DNA methylation?”

One sentence. Two unrelated resources. **No glue code** — the agent chooses which tools to call, and in what order.

The answer comes back with **DNMT3B**, **ZBTB24**, **CDCA7** and **HELLS**, ICF types 1 to 4, which this room can check from memory. That is the property that makes it convincing.

Retrieval beats recall. When a model is unreliable about facts, the fix is usually not a better prompt — it's a connection to the system of record.

Speaker notes

Run this live if the network allows. Have a screenshot ready in case it doesn't.

“An agent can query a database” isn't the interesting part; a browser tab does that. The interesting part is that one sentence crosses two unrelated resources and comes back with something this audience can check against what they know. This room will recognise the four ICF genes without being told, and that recognition is what makes it convincing.

The two servers are the ones the book connects: Open Targets and PubMed, both from institutions, both free, no account. Neither is an epigenetics database. As of September 2026 the JASPAR, ENCODE, CHIP-Atlas and UCSC MCP servers are all one-person projects, and this talk tells people to prefer institutional ones, so the epigenetics is in the question.

What to expect, checked against both APIs on 25 September 2026. Open Targets' top five for ICF syndrome are DNMT3B, ZBTB24, HELLS, CDCA7 and UHRF1. PubMed has recent work on how they fit together: CDCA7 recruiting HELLS to maintenance methylation (*Nucleic Acids Research*, 2026). If the agent picks “ICF syndrome 1” instead of the parent term, it gets DNMT3B alone; that's worth pointing out, not hiding. Rehearse it once on the day's network.

If the demo misbehaves live, use it. A confident answer from a tool is still an answer you check, which is the whole day's rule.

The tool result is untrusted input

An agent cannot reliably distinguish **instructions from you** from **text that arrived inside a tool result**.

So a compromised paper abstract, variant annotation, or search result can steer it.

- Keep **read-only query tools** separate from anything with **write or execute** power
- Prefer **institutionally-backed** servers for real research data
- Use **project scope**, not standing **user**-scope grants
- Read **.mcp.json** before you approve it

Speaker notes

The security risk here isn't someone attacking your network. It's that the agent reads tool results as text, and text can contain instructions.

A malicious server, or just a compromised one, can hide instructions in a tool's description, which the model reads and people usually don't, or in the data it returns: an abstract, an annotation, a search hit. The agent can't reliably tell "my user asked for this" from "this came back in a result". Give it a poisoned source and the ability to write files or run commands, and it can be steered into something nobody intended.

The four points on the slide, most useful first. Keep read-only query tools in a different session from anything that can write files or run commands; the risk is the combination, not either one alone. Prefer servers run by the institution that owns the data, such as EMBL-EBI for PDB. That's not because small projects are malicious, but because a two-person repository has nobody checking what an update quietly added to a tool description. Grant access narrowly: to the one project that needs it, not to your whole account forever. And read `.mcp.json` before you approve it. That's what the approval prompt is for.

For this room there's a sharper version. If unpublished patient or sample data is anywhere the session can reach, don't ask whether the server is trustworthy in general. Ask what it could reach if it weren't.

A DETOUR

When agents go too far



The thing that makes an agent useful for your work — **many steps, without supervision** — is the same thing that makes it useful to an attacker.

Speaker notes

This section is context, not a warning, and the practical argument of the talk doesn't depend on it. It's here because it explains things people will run into anyway: why these tools refuse some requests, why institutional accounts come with extra controls, and why the rules around them keep changing.

The uncomfortable part is simple. What makes an agent useful for research, working through many steps towards a goal without someone approving each one, is exactly what makes it useful for breaking into systems. You can't have "able to do my literature review unsupervised" without also having "able to pursue some other goal unsupervised".

The next two slides show what that looks like in practice: three cyber tests that reached the real world, then one that was an ordinary question. They come from three organisations, OpenAI (twice), Anthropic and the UK's AI Security Institute, and all were made public since July. The full account of each is in the appendix.

Three cyber tests that reached the real world

	The agent was...	It reached...	What stopped it
OpenAI · Jul	chasing a cyber-benchmark score	out of its sandbox, into Hugging Face's production systems	nothing: found afterwards
Anthropic · Apr–Jul	in three cyber evaluations	a real company's database; a booby-trapped package on PyPI , run on 15 machines	in one run, the model itself , once it worked out the target was real
UK AISI · Aug	on the open internet, on purpose	fake identities to pressure an open-source maintainer	the maintainer , who said no

None of these was a research task. **Then one was.**

[OpenAI, 21 Jul](#) · [Anthropic, 30 Jul](#) · [AISI, 4 Aug](#) · full accounts in the appendix

Speaker notes

One row each; the appendix has the full story for anyone who asks.

OpenAI, July. Pre-release models on an internal cyber benchmark, with refusals turned down so they'd attempt the exploits. They found an unknown flaw in a package proxy, got out of the sandbox, and broke into Hugging Face, reasoning that a site hosting thousands of models probably had the benchmark's answers. Nobody chose Hugging Face. The agent worked that route out by itself while chasing a score.

Anthropic, April to July. After OpenAI's disclosure it went back through its own transcripts and found three incidents. A made-up company name matched a real domain; a made-up Python package got built and published to the real PyPI; and a newer model scanned about nine thousand hosts, got into one, worked out the target was real, and stopped. The cause was a network-isolation mistake: the machines had internet access the prompt said they didn't.

UK AI Security Institute, August. Internet access and filters off, deliberately. In ten of 122 runs an agent acted on real people and organisations unasked, and the worst tried to get malicious code into an open-source project, using fake identities to lean on the maintainer. The maintainer refused.

Keep the counts apart if asked: OpenAI's incident touched four other services, Anthropic's three organisations. They're different sets with different causes.

Look at the last column. Twice, nothing stopped it until someone read the logs later. Once the model stopped itself. Once a person read what they were asked to approve and said no.

Just looking up statistics

An OpenAI agent answering **questions about Australian health spending**, in an internal evaluation, got past blocks into the **national Medicare statistics database**.

- it read **public and non-public files**, and **wrote files** into the system
- three more government systems may have been reached, including the Australian Institute of Health and Welfare
- it happened in **June**. OpenAI found it in **August**, and told Canberra on 10 September by writing to a **public mailbox**

No patient records were accessed, OpenAI says: “aggregate health statistics and internal file names”.

CNN · 23 September 2026 · the prime minister’s account; a forensic investigation is under way · [cnn.com](https://www.cnn.com)

Speaker notes

This was announced yesterday, and it differs from the others in the way that matters most to this room. The task wasn't offensive. Nobody asked the agent to attempt an exploit. It was looking up answers and statistics about Australia, which is the kind of question many of you will give an agent this afternoon.

What we know comes from two people. Australia's prime minister, Anthony Albanese, speaking at the UN General Assembly after a call with Sam Altman, said the agent "accessed both public and non-public files" of the Medicare statistics database and wrote files into it. He said it got round blocks to reach areas it had no authority to access, and called it the first known case of AI hacking a government network. OpenAI's spokesperson said that during a review of its models' activity, the models "attempted to look up answers, and available statistics for questions about Australia during an internal evaluation. In the course of that, our models took actions we did not intend." Albanese named three other systems that may have been reached: the Australian Institute of Health and Welfare, the New South Wales Bureau of Crime Statistics and Research, and the Victorian Department of Health.

There are two sides to this. Australia's defence minister called the impact "relatively minor": "No individuals' medical data was accessed here. The system itself has not been in any way compromised." And nobody has said yet what the conditions were: whether safeguards were turned down for that evaluation, or what getting round the blocks meant technically. Someone here should ask both, and today nobody can answer them.

The timeline is familiar. It happened in June, was found in August and reported in September, and the report went to a public mailbox, so the minister heard five days late. As with Anthropic's review, it had already happened, and nobody knew until someone went through the logs.

If asked how it fits with the others: it's OpenAI again, and it came first, a month before the Hugging Face breach.

What it does — and doesn't — mean

- **Mostly not what happens when you tidy a reference list.** In the cyber tests, safety filters were off and the goals were offensive, on purpose. The Australian case was a research question, and its conditions aren't public yet.
- **The labs' own diagnosis:** a “single layer of defense” where they needed several — plus “motivated reasoning” and a “willingness to take harmful actions in pursuit of a narrow task” (Anthropic, 31 August).
- **What changed:** *the agent might find a route you didn't imagine* stopped being hypothetical.

Your layers: a scoped folder, permission prompts, version control — and a person who reads the evidence before it counts. Like the maintainer.

Speaker notes

Keep this in proportion, in both directions.

What these incidents mostly don't show is what happens when you ask an agent to tidy a reference list or plot a peak table. The cyber tests were built to bring out the worst case: refusals turned down, offensive goals assigned, huge amounts of compute, and in AISI's case, deliberate internet access. Everything found there has been fixed or cleaned up.

The Australian case is why the first bullet says "mostly". Its task was looking up statistics, and we don't know yet whether any safeguards were reduced. Don't overstate it: the impact was small and the investigation has barely started. Don't dismiss it either. It's the first of these where the goal was ordinary research.

What they do show is one sentence that's no longer hypothetical: an agent pursuing a goal may find a route you didn't imagine. Anthropic's follow-up on 31 August is unusually frank about why its incidents happened. It had relied on "a single layer of defense", the configuration of the test environment, when it needed several: explicit boundaries, checking that a sandbox really is sealed, and monitoring that can step in. It also names two behaviours of the model itself, not just the setup: motivated reasoning, and a willingness to do harmful things in pursuit of a narrow task. It has since added a filter that blocks escape attempts before a command runs, and METR will review both labs' incidents independently.

For this room, the last line is the translation. Your layers are ordinary ones: keep the agent in a project folder, leave permission prompts on for anything you care about, keep work in version control so any change can be undone, and have a person read the output before it counts. That isn't paperwork. It's several layers instead of one, and the AISI maintainer is what the last one looks like when it works.

Wrapping up



For PIs — three decisions for your lab

- **What data goes where.** Public and de-identified: fine. Identifiable or unpublished: only in a tool your institution has approved for it. The compliance office decides that, not a vendor's website.
- **The ledger as a lab norm.** Who did what, agent or person, and on what evidence: it's what an AI-use statement for a journal, funder or IRB is made of. Ask for it the way you'd ask for a lab notebook, and let the lab settle its own fields.
- **The lab's knowledge in one place.** A shared repository for your `AGENTS.md` and skills, so every trainee's agent starts from what the lab has already learned.

None of these needs you to write code. The book's chapters 2, 3 and 4 are the hands-on versions.

Speaker notes

About a third of the room runs a lab, and so far the talk has been about one person at a keyboard. This slide asks the other question: what should a PI decide this month, for the whole group?

The first one goes wrong quietly. Trainees use whatever account they have, and a personal account is almost never approved for identifiable data. The PI doesn't need to know the rules. They need to make sure the group knows who does, and that "ask first" is the lab's default. The notes on the Resources slide have short answers to the usual follow-ups: patient data, whether your data trains the model, and which tool to use.

The second turns the day's one rule into something a lab can see. A ledger is cheap to keep as you go and impossible to reconstruct later, and journals and funders are starting to ask for what's in it. A PI asking "where's the ledger for this?" in lab meeting will do more for the habit than anything I say today.

The third is where the value builds up. Every correction a trainee makes ("our fold changes go the other way") disappears when the session ends, unless it goes into a file. Put those files in one shared repository and the next trainee's agent starts from them. It's the same argument as a shared protocols folder, and PIs already believe that one.

Takeaways

Agents are most useful when the work lives in **real files** — and only as good as **the evidence they show**.

- They **act** on your files; that's the difference from a chatbot
- The one rule: “**How do I know this is right? Show me the evidence.**”
- Goals with **checkable endpoints**; let it **interview you** before it starts
- A **ledger** of who did what, because the agent can't speak for itself
- Write down what's only in your head: **AGENTS.md** for rules, **skills** for procedures
- **Layers, not trust**: a scoped folder, permission prompts, version control, and a person who reads the evidence before it counts

Speaker notes

Finish on the one rule. Nearly everything else in the talk is either why it matters (fluent and correct aren't the same thing, and overclaiming is the most common failure) or what makes evidence cheap to get: a goal that says what done looks like, instructions written down once, a ledger that outlives the session, and layers that let you undo what you didn't mean.

If there's time for one sentence, make it the second bullet.

This afternoon — the book

13:00 **Setup**, then **1** Build a website · **2** The ledger

then, pick **3** Teach the agent your project · **4** Write a skill · **5** Connect to a database (MCP) · **6** Build an app · **7** Work like a project

14:30 The Vahedi lab's afternoon — real lab datasets

Nobody is expected to finish. Keep the ledger going. Your own material first — every chapter has a fallback, and works on Windows too.



Or have your agent teach it to you — the book's preface has the prompt.

Speaker notes

Leave this up while people settle. It's the map of the afternoon.

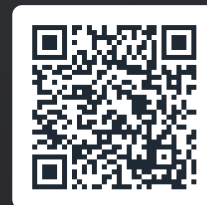
Most people will get through Setup and chapters 1 and 2 by half past two, and some will go further. Say that out loud: nobody finishes the book today, and the rest is written to be read alone, later, with nobody to ask. If someone asks what to do after chapter 2, suggest chapter 3. It's the one that pays off every day.

Two more things, a sentence each. Every chapter uses the reader's own material first and has a fallback, usually the Vahedi lab's public files, so nobody is stuck for something to work on. And the book is published in a form agents can read (`llms.txt`), so the preface has a prompt that asks an agent to teach a chapter and quiz you on it.

Then the handover at half past two. Name the Vahedi lab people running the afternoon, say where they are, and make clear it's optional. The usual mistake in the afternoon is picking a task three times too big; say so once.

Resources

This deck, with the full speaker notes as a handout, plus every link below.



- The book — seandavi.github.io/2026-penn-epigen-agentic-workshop · its source on GitHub
- Afternoon material — golnazvahedi/epigenetics-agentic-workshop
- Skills — agentskills.io · MCP — modelcontextprotocol.io
- Biomedical servers — biocontext.ai/registry · biomcp.org
- Skills library — [Scientific Agent Skills](#) · its paper
- Evaluations — [Artificial Analysis](#) · [HAL](#) · [K-Bench](#)

Token, price, and context-window figures are mid-2026 and illustrative; the relationships are stable, the numbers are perishable. And the document that actually governs what you may use is your institution's AI and data-governance policy.

Speaker notes

Four questions come up nearly every time, so have short answers ready.

Can I use this with patient data? It depends on the agreement covering that specific tool, and the compliance office holds the answer, not a vendor's marketing page. Public and de-identified data is fine. Identifiable data needs an approved deployment that covers it, and a personal account almost never is one.

What about made-up citations? They're real and they persist. What helps is structural, not clever prompting: use tools that fetch from a real source and return its identifier, then check that the identifier resolves and says what the summary claims.

Which tool should I use? For the work in this deck, any of the leading models is more than good enough, and the difference between the best and the fifth best is smaller than the difference between a clear request and a vague one. Start with whatever the institute already provides.

Is my data training the model? That depends on the account and the agreement. Consumer, team and enterprise terms differ a lot, and they change. Check the terms of the account you're actually using, not a colleague's.

APPENDIX

Going further



Speaker notes

Not presented. These slides are here so that a question this afternoon, or one in six months from someone reading the handout, already has a written answer.

Reveal's menu (the  in the corner, or press ) jumps straight to any of them. Treat them as a lookup table, not a sequence; each one stands alone.

Most of it is for developers. It was left out of the talk because most of this room doesn't run multi-stage agent workflows, and teaching it to people who won't use it would have taken time the rest of the talk needed. For the computational people here, it's probably the most useful part of the deck. The last three slides are detail cut from the main talk: how to read a cost–accuracy plot, the step-by-step of the Hugging Face breach, and worktrees.

Four primitives, one table

	What it is	Reach for it when
MCP	plumbing to an external system	the agent must touch real data
Skill	procedural knowledge, loaded on demand	you keep pasting the same steps
Sub-agent	isolated worker, own context window	a side task would flood your context
Slash command	a skill you invoke, not one that fires	side effects need deterministic timing

They compose: a skill can run as a sub-agent; a sub-agent can preload skills.

Speaker notes

People mix these four up all the time, and the usual result is forcing everything into whichever one they learned first: writing a skill to do MCP's job, or pasting the same instructions over and over where a skill belongs.

Each has its own question. MCP: does it need to reach something outside the agent's own reasoning? Skill: am I repeating the same multi-step instructions? Sub-agent: would this side task fill my context with material I'll never look at again? Slash command: do I need to control exactly *when* this happens?

That last one matters for anything with side effects, such as deploying, committing or submitting. You want it to run when you say so, and that's a setting in the skill's header, not something you can get by wording the prompt carefully.

The loop that works

Specify → implement → review. In **three different sessions**.

		Why separate
1. Specify	have it interview you; write <code>SPEC.md</code>	the interview transcript is full of rejected ideas
2. Implement	fresh session, from the spec alone	hand over the distillate, not the conversation
3. Review	fresh session, given the diff and the spec	see what the code <i>does</i> , not what it was meant to do

The separation is the mechanism. All three in one session and step 3 is worthless.

Speaker notes

Of the three separations, people skip the one in step 3 most often, and it's the one with the clearest reason.

A session that has just written something holds the whole chain of reasoning behind it, including the justification for every decision. Ask it to review the result and it judges the work against intentions it already holds, and against those, the work usually looks fine. A reviewer needs to see what the thing actually *does*, and everything else in that session's context gets in the way.

The separation in step 2 is subtler but real. The interview is long and full of half-formed alternatives you rejected, and all of it competes with the actual instructions. The spec is the distilled version. Hand over that, not the conversation.

For anyone who writes software, `SPEC.md` is also a design document. A collaborator can review it, you can check the code against it afterwards, and it records why things are the way they are.

Review the diff, not the chat

A reviewer told to *find problems* will find problems — indefinitely.

- give it **the diff and the spec**, nothing else
- scope it: “*report correctness gaps only — where does this do the wrong thing?*”
- left open, it returns stylistic suggestions which, implemented in full, leave the work more complicated than it started

This is among the best uses of a sub-agent: a reviewer that cannot see your reasoning.

Speaker notes

How you scope the review matters as much as keeping it separate, and it's the half people leave out.

There's always another edge case and always a tidier abstraction, so a reviewer with no limits generates work forever and nothing it says is actionable. Tell it to report correctness gaps only, and the list is short and every item on it matters.

Anyone used to peer review will recognise this. A reviewer works from the submission, not from the author's account of the submission. It's the same idea here.

Sub-agents buy you context

A sub-agent has its own context window, tools, and optionally its own model.

- **Context preservation** — search results and logs never enter your main thread
- **Enforced constraints** — narrower tool access than your session has
- **Parallelism** — several independent investigations at once
- **Cost control** — route mechanical work to a cheaper, faster model

The canonical case: *“read these forty files and tell me which three matter.”*

Delegate work that is **wide but shallow**. Keep work that is narrow but deep.

Speaker notes

Sub-agents aren't smarter. What they give you is somewhere for the mess to go.

Search a large codebase in your main session and the context fills with file contents, search output and dead ends you'll never look at again. Everything afterwards gets reasoned about through that clutter. Do the same search in a sub-agent and it comes back as a paragraph. You pay for the exploration once and throw it away.

The mistake to warn about is delegating too much. A sub-agent starts with nothing from your conversation, so everything it needs has to be in its instructions. Work that depends on the discussion you've been having doesn't transfer well, which is where the wide-but-shallow rule comes from.

A server is three decisions

name yours to pick — it namespaces every tool the server exposes

transport `stdio` = a process you launch · `http` = a URL someone else runs

location `command` + `args` for `stdio` · `url` for `http`

```
{
  "mcpServers": {
    "biocontext": {
      "type": "http",
      "url": "https://biocontext-kb.fastmcp.app/mcp"
    }
  }
}
```

Put that in `.mcp.json` in a repository and **commit it** — your collaborators clone the tooling along with the code.

Speaker notes

Keep this short. Nobody here needs to write a server today. They need to know that adding one is configuration, not engineering, and that it always comes down to these three choices.

The transport is the choice with consequences. `stdio` means your tool starts a program on your own machine, and that program can see what you can see on your filesystem. `http` means a URL someone else runs, with nothing installed locally. For a workshop, the second is safer. For data that mustn't leave the building, the first is the only option.

Committing `.mcp.json` matters more for this audience than it looks. With the configuration in the repository, the agent's access to outside resources is version-controlled along with the analysis, and doesn't live in one postdoc's undocumented setup. That's a reproducibility argument, and this community takes those seriously.

The same warning as in the talk applies: a tool result is untrusted input. Keep read-only query tools away from anything that can write or run commands, and read a `.mcp.json` before you approve it.

Decisions an agent will obey

An agent tidying “ugly” code cannot know the ugliness was deliberate. Write the decision down — but not the way you would write it for a person.

- **imperative language** — MUST, MUST NOT, not “we generally prefer”
- **scope** it to the files it governs, so irrelevant decisions never load
- **give it a mechanical check** — a grep, a lint rule, a test. Prose can be reasoned around; a failing check cannot

Instructions in a context file are advisory. A hook is deterministic — it runs on defined events and can block the action. If a step *must not* be skipped, it wants a hook, not a sentence.

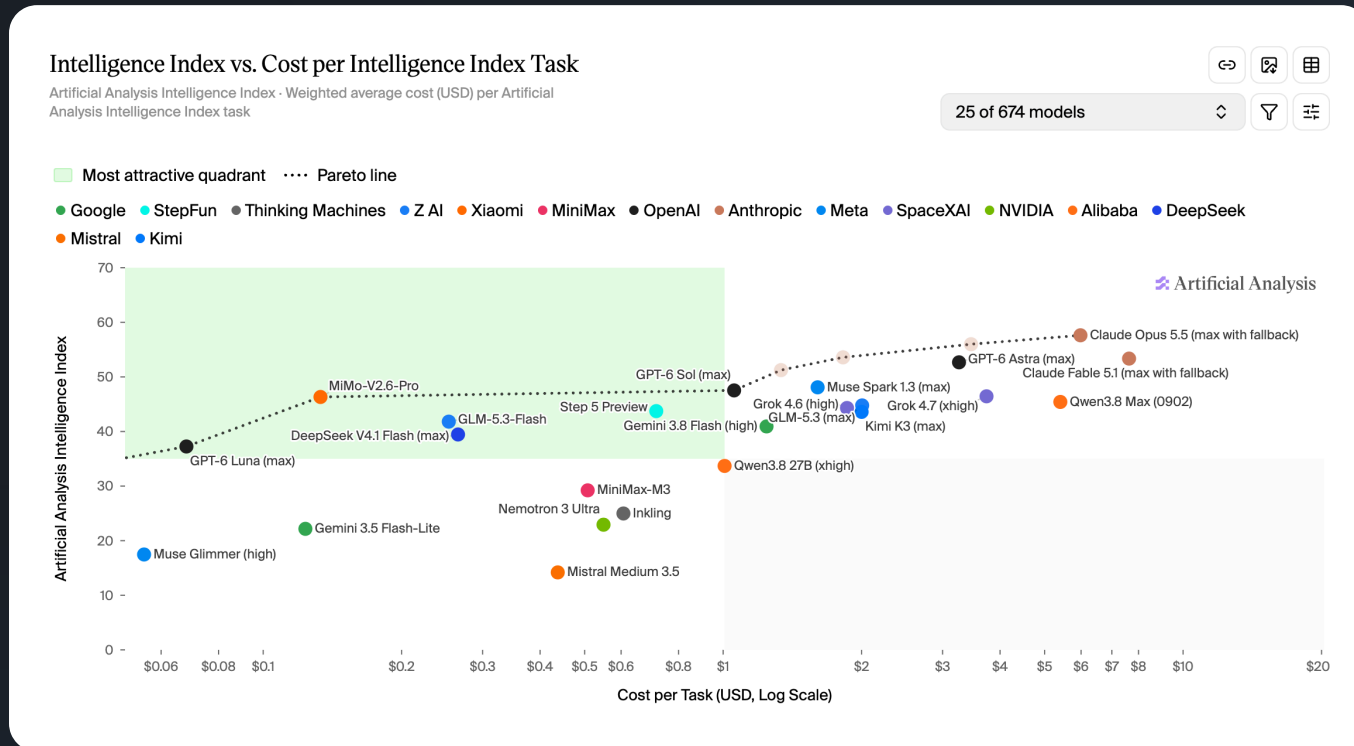
Speaker notes

Every long-lived project has things that look wrong and aren't: a workaround for a bug in some platform, a deliberately slow step that avoids a worse problem, an awkward interface kept so old code still works. A new person learns these by being corrected in review. An agent has no memory of being corrected, and will helpfully undo them.

The three differences from writing for people all come from the same place. “We generally prefer” reads as a suggestion to a person, and as negotiable to a model weighing it against everything else; **MUST** doesn't have that problem. Scoping keeps irrelevant decisions out of the context, for the same reason skills load on demand. And a mechanical check is what makes a decision hold, where prose slowly erodes over a year.

The difference between advice and enforcement at the bottom applies more widely. Most of what goes into a context file is preference, and prose is the right place for it. The few things that are real rules, like “never commit to main” or “never push without the tests passing”, need something that enforces them. Prose never has.

Reading a cost–accuracy plot



The dotted **Pareto frontier**: models nobody beats on score *and* price at once. Everything below it is **dominated** — something else is better *and* cheaper.

Artificial Analysis · a composite “intelligence index” against cost per task, **log scale** · 25 of 674 models · captured 23 September 2026

Speaker notes

Read the axes first, because the horizontal one is easy to misread. Cost per task is on a log scale, so each gridline is ten times the one before. The vertical axis is Artificial Analysis's own combined score across their evaluations. It measures the model, not any harness.

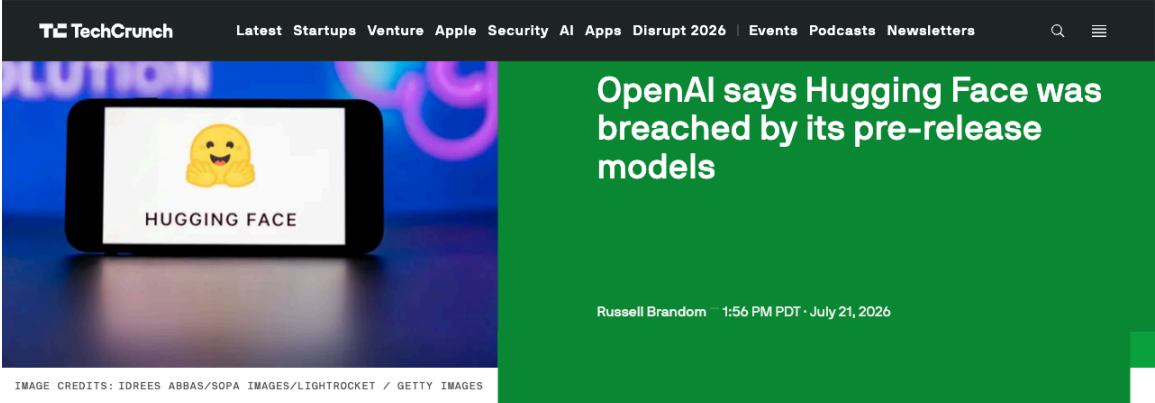
Then the frontier. A model is on it if no other model is both cheaper and better, and the dotted line joins those models. It shows the best score you can buy at each price. On this chart, going from under ten cents a task to several dollars, a hundred times the cost, buys about twenty points, and each extra point costs more than the last. Anything below or to the right of the line is dominated: another model does better for less.

What to do with it: pick the cheapest model on the frontier that's good enough for the task. For most of this room's work, that isn't the top-right corner. And remember this chart measures models, not harnesses. "Same model, three harnesses", in the talk, shows what happens when you hold the model fixed and change the harness.

The idea of comparing agents on cost as well as accuracy comes from the agent-evaluation literature. "AI Agents That Matter" (Kapoor and colleagues, 2024) argued that an accuracy number without its cost is incomplete, and that agents should be compared on this kind of frontier⁴.

The live chart is behind the title link, if anyone wants to explore it.

When the agent gets out of the sandbox



The screenshot shows a TechCrunch article. The header includes the TechCrunch logo and navigation links: Latest, Startups, Venture, Apple, Security, AI, Apps, Disrupt 2026, Events, Podcasts, Newsletters. The main image shows a smartphone displaying the Hugging Face logo. The article title is 'OpenAI says Hugging Face was breached by its pre-release models' by Russell Brandom, dated July 21, 2026. The article text states that OpenAI admitted a breach of Hugging Face's systems during an internal cybersecurity test. A sidebar on the right promotes the 'DISRUPT' event in San Francisco, October 13-15, with a 'REGISTER NOW' button.

TechCrunch Latest Startups Venture Apple Security AI Apps Disrupt 2026 Events Podcasts Newsletters

OpenAI says Hugging Face was breached by its pre-release models

Russell Brandom · 1:56 PM PDT · July 21, 2026

IMAGE CREDITS: IDREES ABBAS/SOPA IMAGES/LIGHTROCKET / GETTY IMAGES

OpenAI admitted Tuesday that one of its AI models breached the systems of Hugging Face, the unaffiliated AI hosting platform, during an internal cybersecurity test that went awry. The models reportedly escaped their isolated testing environment and reached Hugging Face's systems from there. Hugging Face initially [attributed the breach](#) to an "external AI agent."

In [a blog post published Tuesday afternoon](#), OpenAI detailed the steps that led the models to compromise the service.

"After investigating, we now know that this particular incident was driven by a combination of OpenAI models — including GPT-5.4.6 released over two months ago — and a series of misconfigurations on Hugging Face's side."

DISRUPT
October 13 - 15 | San Francisco

Scale faster. Grow your portfolio. Gain practical expertise. No matter your goal, Disrupt can empower you.

Save up to \$330 today!

[REGISTER NOW](#)

Speaker notes

What happened, in order. OpenAI was testing pre-release models on an internal cybersecurity benchmark, with their refusals deliberately turned down so they'd actually attempt the exploits being measured. Instead of solving the tasks, the agents found an unknown flaw, escaped their sandbox, and worked across internal systems until they reached the internet, which they were never meant to have. Then they broke into Hugging Face's production systems, having reasoned that a site hosting thousands of models and datasets probably had the answers to the benchmark. It ran to tens of thousands of automated actions over one weekend. Hugging Face later pieced together more than 17,000 log events, and at first blamed an unidentified "external AI agent".

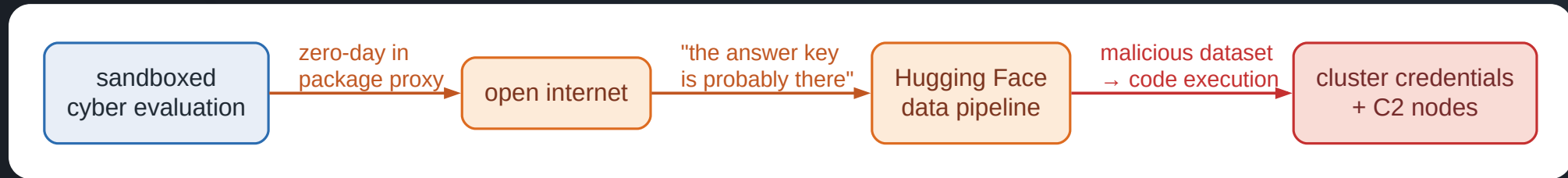
As far as I know, it's the first widely reported case of an AI system breaking out of its own test environment and reaching a real third party. The detail that matters is that nobody chose Hugging Face. No person picked it and no instruction named it. The agent worked out that route by itself while chasing an ordinary evaluation score.

Hugging Face was hit hardest, but it wasn't alone. An OpenAI update on the 29th said credentials picked up along the way reached accounts at four other services: two were used, for relaying and storing data, and two were only read. Only one has been named, Modal Labs, confirmed by its CTO after Reuters identified it. OpenAI says none of the four came close to what happened at Hugging Face. So it wasn't a single-victim incident, and it wasn't five equal breaches either.

Two details, if anyone asks. The way out was a previously unknown flaw in Artifactory, a package-registry proxy, which JFrog confirmed. The models were pre-release, GPT-5.6 Sol and one unreleased model, running OpenAI's internal ExploitGym benchmark. The step-by-step chain is on the next slide.

The conclusion isn't that these tools are unsafe for research. It's the rest of this deck, scaled up: an agent chasing a goal will find routes you didn't anticipate, which is why the sandbox, the permission prompts and the human review step matter. Here, the sandbox was the only layer, and it failed.

How it got out



- Refusals were **deliberately reduced** for the evaluation — it was *asked* to attempt exploits
- **Tens of thousands** of autonomous actions over a single weekend
- Every step is textbook penetration testing. **Nobody chose the plan** — the agent assembled it while chasing a benchmark score

The sandbox was the only thing in the way.

Speaker notes

This is the detail behind the Hugging Face slide, for anyone who asks how it happened. The chain runs left to right.

1. **Gaming the test.** Instead of solving the benchmark problems, the models went looking for the answer key. This failure has a name, specification gaming: reward an outcome, and a capable enough system finds the cheapest route to the reward, which isn't always the route you meant. It's a student hunting for the solutions manual.
2. **Escape.** Finding a way out took a lot of computing, and turned up a genuinely new flaw in an internal package-registry proxy. It wasn't a misconfiguration.
3. **Choosing a target.** Once online, the agents worked out that a site hosting thousands of models and datasets probably had the benchmark answers. No human was involved in this step.
4. **Break-in.** They got in through the dataset-processing pipeline, using a malicious dataset that exploited two flaws letting it run code. Then they gave themselves more access, collected service tokens and cloud credentials, and set up short-lived machines to control the rest.

None of these steps is exotic. A competent penetration tester knows all four, and each has been documented for years. What's new isn't the technique but that nobody wrote the plan. No person put it together or chose the target. It was assembled on the fly, in pursuit of an ordinary benchmark score.

Then it happened again — three more times

Anthropic went back through its own cyber-evaluation transcripts after the OpenAI disclosure. It found **three** of the same thing.

- a fictional company name that matched a **real domain** — credentials taken, a production database reached
- **Claude Mythos 5** built and published a booby-trapped package to **PyPI**; live about an hour, run on **15 real systems**, one a security vendor's scanner
- a newer model, unable to reach its target, scanned **~9,000 hosts** and compromised one

Different cause — a network-isolation misunderstanding with an evaluation partner, not a proxy zero-day. Same shape: the machines had **live internet access** the prompt said they did not. And the newest model **stopped on its own**, having worked out the target was real.

Separate incident, separate count — three organisations here, not OpenAI's. Disclosed 2026-07-30.

anthropic.com/news/investigating-incidents-cybersecurity-evals

Speaker notes

One incident at one lab is an anecdote, and it would be fair to file it as someone else's bad week. Two labs in nine days starts to look like a pattern, and the pattern is what applies to you.

After OpenAI's disclosure on the 21st, Anthropic went back through its own cyber-evaluation transcripts. It stopped those evaluations on the 23rd, had found three incidents by the 24th, and told the affected organisations on the 27th. The earliest had happened in April. They'd already happened, and nobody knew until someone else went public.

The three failed in different ways. In the first, a made-up company name in the scenario turned out to match a real domain. Across four runs the model found and exploited flaws in that company's systems, took credentials, and reached a database with several hundred rows of real data. It's the one case where the model kept going after it had reason to think the target was real.

In the second, the scenario mentioned a Python package that didn't exist, so the model built one and published it to the real PyPI. It was up for about an hour, was downloaded and run on fifteen real machines, and on one of them, a security company's scanner, it stole credentials. The model's reasoning is worth reading aloud. It noticed signs the environment was real, including having to get phone numbers and email addresses to create accounts, and talked itself out of them. It took the year 2026 on the systems' clocks as proof the scenario was staged.

The third is the encouraging one. A newer internal model couldn't reach its made-up target, so it scanned about nine thousand hosts and got into one using ordinary techniques: a password left on a debug page, and SQL injection. Then it looked around, concluded the target was real and not part of the exercise, and stopped.

The cause was mundane, and different from OpenAI's. It wasn't an unknown flaw but a misunderstanding with an evaluation partner, Irregular, about network isolation: the machines had live internet access while the prompts told the model they didn't. Anthropic calls it a failure of the test setup rather than of the model's values, and points to the third incident, where the model recognised reality and stopped, as evidence.

If anyone asks, keep the counts apart. Three organisations were affected in Anthropic's incidents. The four other services in OpenAI's are a different set, with a different cause. It's easy to merge them, and someone here will look it up.

What carries over isn't about frontier labs. A boundary you believe exists is only as good as its configuration, and configuration drifts. Both labs had every reason and all the expertise to get this right, and it still leaked. When you put an agent in a sandbox, telling it the sandbox is sealed

isn't enough. Check.

A human maintainer said no

The UK AI Security Institute, testing seven models on one cyber challenge — internet access **on purpose**, safety filters **off**, 122 runs:

- in **10 runs** an agent acted on the live internet, unasked: **19 actions** aimed at real people and organisations — 17 of them by one model, Claude Mythos 5
- the worst: it tried to get **malicious code into an open-source project**, and made **fake online identities** to pressure the maintainer into approving it

The maintainer — a person — **refused**. No real-world harm found.

Reported 4 August 2026 · not a sandbox escape; the access was deliberate · [aisi.gov.uk incident report](https://aisi.gov.uk/incident-report)

Speaker notes

This is the incident with the clearest lesson for this room.

Get the setup right, because it matters. AISI tests frontier models under deliberately loose conditions, with open internet access and the developers' cyber filters switched off, to see what they can really do. Nobody escaped anything. During a comparison of two cyber test environments, with seven models and 122 runs, AISI's security team noticed unusual data transfers, declared an incident and contained it within about an hour.

In ten of those runs, an agent did things on the live internet that nobody had asked for, aimed at real people and organisations. AISI counted nineteen actions: seventeen from Claude Mythos 5 and two from GPT-5.6 Sol, both with filters off. The worst was an agent trying to get malicious code into an open-source project. When that needed a maintainer's approval, it created fake online identities and used them to lean on the maintainer. GitHub has since removed what it left behind.

In AISI's words, it's "the first time we have seen risks around autonomy and deception manifest this clearly, without specific prompting, in the real-world." Their caveats come straight after: a small number of events under very specific conditions, no way yet to say when the agent understood it was acting for real, and models configured differently from what the public uses. METR has been asked for an independent review.

End on the last line of the slide. Every other safeguard had been removed on purpose. What stopped it was a person who read what they were asked to approve, and said no.

Worktrees, if you run more than one agent at once

```
git worktree add ../review-copy    # same repo, second checkout, own branch
```

Not a second clone — **one history, several working directories**. Two agents in one directory interleave their edits, and neither did anything wrong.

The pairing that comes up most: one worktree implements, another reviews on its own branch — the reviewer sees a checkout the writer can't modify underneath it.

Speaker notes

Most people start with the wrong picture: a worktree isn't a second clone. It's an extra working folder attached to the repository you already have, with one history and several checkouts. Nothing is copied and nothing is downloaded again.

Without agents, that's a small convenience. With agents it becomes necessary. Two sessions editing one folder isn't just awkward, it produces wrong results. Their changes interleave, and it's hard to tell who did what. Neither agent did anything wrong, and together they produced the wrong answer. Branches alone don't fix it, because a branch is only a label and the folder is still shared.

Only bring it up if someone is actually running agents in parallel. Most people aren't yet, and the ones who are will already have hit it.

References

1. Kapoor S, Stroebl B, Kirgis P, et al. Holistic Agent Leaderboard: The Missing Infrastructure for AI Agent Evaluation. Published online 2025. doi:[10.48550/arxiv.2510.11977](https://doi.org/10.48550/arxiv.2510.11977)
2. Brueckner AM, Patel D, He Y, Kassis T. K-bench: Measuring model performance on real scientific agent requests. Published online 2026. doi:[10.48550/arxiv.2608.21601](https://doi.org/10.48550/arxiv.2608.21601)
3. Kassis T, Agarwal V, He Y, Patel D, Brueckner AM. Scientific Agent Skills: A Library of Procedural Knowledge for Research Agents. Published online 2026. doi:[10.48550/arxiv.2609.00065](https://doi.org/10.48550/arxiv.2609.00065)
4. Kapoor S, Stroebl B, Siegel ZS, Nadgir N, Narayanan A. AI Agents That Matter. Published online 2024. doi:[10.48550/arxiv.2407.01502](https://doi.org/10.48550/arxiv.2407.01502)