

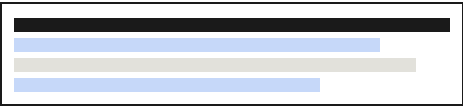
START HERE

How AI works, end to end

What actually happens between pressing Enter and reading the reply — and why AI behaves the way it does.

PART ONE

What the model is



A chat is a list of messages in a fixed-size box. It has no memory of its own.

PART TWO

What it can't do



It guesses the next word. That's why it invents things, and can't do sums.

PART THREE

How it reaches the world



It can't reach anything itself, so it asks the software around it to fetch things.

PART FOUR

Agents at work



An agent is that same model, run in a loop until the job is done.

THE WORDS YOU'LL HEAR · 1 OF 2 · THE BASICS

LLM (large language model)

The text generator itself. “Large” = trained on an enormous amount of text; “language model” = it models which words tend to follow which. People say “the model” for short.

Token

A chunk of text, roughly four characters. Not quite a word. Everything is measured in these.

Context window

The fixed-size box holding everything the model can see right now, measured in tokens.

Hallucination

When the LLM states something false as if it were fact. Not a glitch – a side-effect of how it generates text.

Prompt

What you send it: the goal, any constraints, and what a good answer looks like.

Messages (the conversation)

The chat as a numbered list of turns – you'll also hear “message history” or, in code, “the messages array”. It's resent from the top every turn.

System prompt

Hidden instructions at the top of that box, written by whoever built the product – not by you. It's why a support bot won't discuss poetry.

Streaming

The reply arriving token by token, which is why it looks like typing.

THE WORDS YOU'LL HEAR · 2 OF 2 · THE MACHINERY AROUND THE MODEL

Harness

The app around the model – it keeps the list, runs the tools, and decides what to send.

Skill

A written playbook for a kind of work. Instructions only – it teaches, it doesn't add powers.

Agent · agent loop

A model run in a loop: understand, look, act, check, repeat – until the goal is met.

Retrieval

Searching your documents and pasting only the relevant passages into the window.

Tool call

The model asking the harness to do something real, then reading the result as text.

Plugin / connector

A package you switch on that gives the harness new tools to offer.

Sub-task · sub-agent

A job split into independent pieces; a helper agent can take one, with its own fresh window.

Persistent artifact

Anything saved outside the window – a file, a spec, a commit – so it survives the forgetting.

THE CONVERSATION IS A LIST OF MESSAGES

SYSTEM PROMPT · HIDDEN

“You are ChefBot. Only discuss cooking — politely decline everything else.”

YOU

“What's a good pasta recipe?”

ASSISTANT

“Try cacio e pepe — three ingredients...”

YOU

“Make it vegetarian?”

↳ every send = this ENTIRE list again, top to bottom

THE CONTEXT WINDOW



Never just your latest message — the *whole list*, from the top, every single time. The model has no memory of its own.

TEXT BECOMES TOKENS BECOMES NUMBERS

1 · YOUR TEXT

The spaghetti was unbelievably good.

2 · CHOPPED INTO TOKENS

The

spag

hetti

was

unbel

ievably

good

.

common words = 1 token · rare words get split

3 · EACH TOKEN IS JUST A NUMBER (IDS BELOW ARE ILLUSTRATIVE)

791

83904

71197

574

65847

98650

1695

13

~4 characters

per token, on average

~750 words

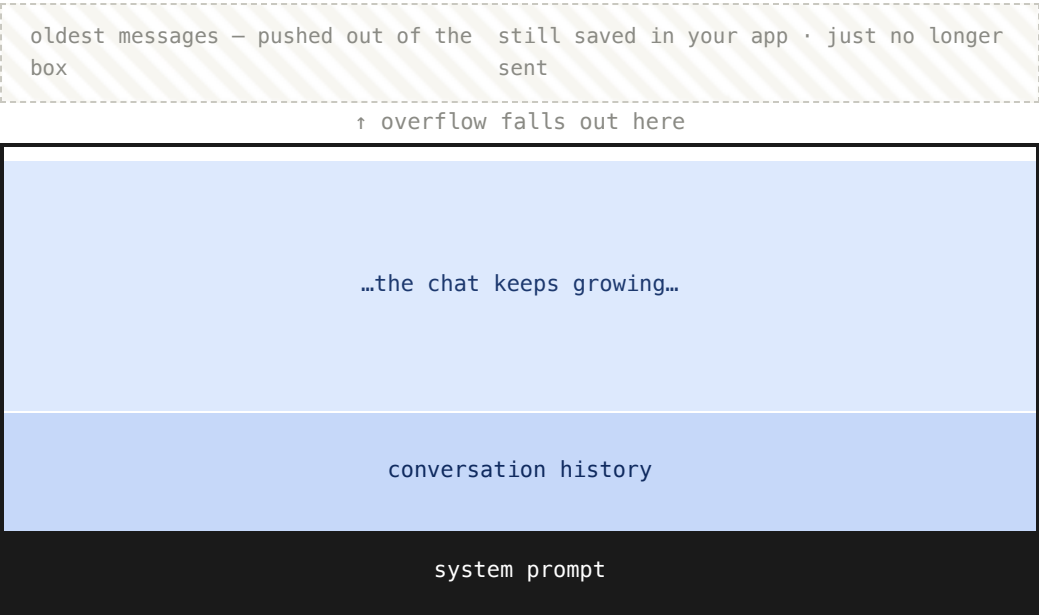
≈ 1,000 tokens of English

\$ / token

pricing and limits are counted in tokens

NOT ALL TEXT IS EQUAL “good” = 1 token · “unbelievably” splits into 2 · and dense code, JSON or logs use roughly twice as many tokens per character as plain prose

THE CONTEXT WINDOW — WORKING MEMORY WITH A HARD LIMIT



199,600 tokens in the box a common limit today: ~200,000 tokens
only what's inside the box is sent to the model — nothing else exists for it

THE KEY IDEA

Once full, the oldest content is dropped —
the model truly forgets it.

WHEN THE BOX FILLS UP

92% full · 40 turns of chat

slower · pricier · easier to distract

1 · Auto-compact

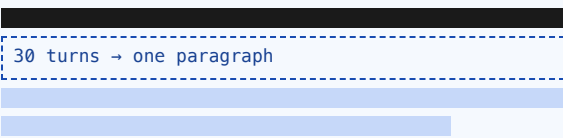
clears the oldest tool results, then summarises



creates a summary of the conversation – detail doesn't survive

2 · Compact

ask the model to summarise the old part, keep the summary



the same thing, but you guide what's worth keeping

3 · Start fresh

new chat, paste back only what matters



cheapest and sharpest – you decide what's worth carrying over

A clean, compact window beats a big messy one: everything in the box competes for the model's attention, so irrelevant history actively makes answers worse.

ANYTHING THAT MUST OUTLIVE THE WINDOW HAS TO BE WRITTEN DOWN



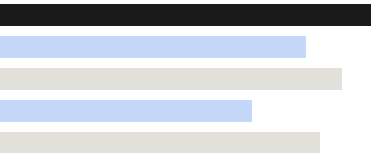
This is why “write the spec first” works so well with agents: the spec is a persistent artifact. It outlives the chat, and it gives you and the agent one shared definition of “done”.

Rule of thumb: if losing it would hurt, it doesn't belong in the chat – it belongs in a file.

EVERY MESSAGE YOU'VE SENT — SHIPPED TO A SEALED-OFF MODEL

Your device

holds the conversation



system prompt + every message
so far

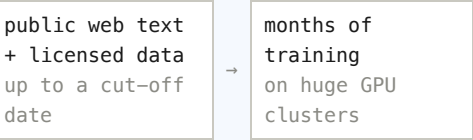
the ENTIRE list →
every message, every time

← the reply streams back,
token by token



A DATACENTER, SOMEWHERE ELSE

ALREADY DONE, LONG AGO



↓ baked in once, then sealed

The model

frozen weights on racks of GPUs — a
sealed box that turns text into text

■■■■■■■■■■ reading all of it, then
predicting one token at a
time...

✓ can see
only the messages
you just sent

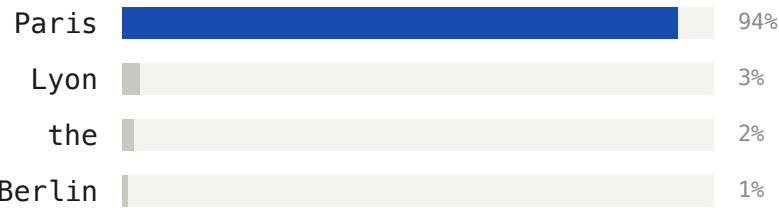
✗ cannot see
the live internet,
your files,
today's news

No memory between turns. No internet. Nothing past its training.

WHY IT SOMETIMES MAKES THINGS UP

“The capital of France is _____”

THE MODEL SCORES EVERY POSSIBLE NEXT TOKEN



it picks a likely one, adds it to the text, and repeats
– one token at a time, forever forward

SO WHAT'S A HALLUCINATION?

When it doesn't know, plausible tokens still score highest —
so out comes a confident, fluent, wrong answer.

There's no built-in “I don't know”. Hence tools and sources.

SO CAN IT DO MATHS? – RECALL VS. CALCULATION

“ $1 + 1 = 2$ ”

✓ right – but not because it added anything.
“ $1 + 1 =$ ” is followed by “2” countless times in its training text, so “2” is overwhelmingly the likeliest next token. This is recall.

“ $4,391 \times 7,208 = 31,648,528$ ”

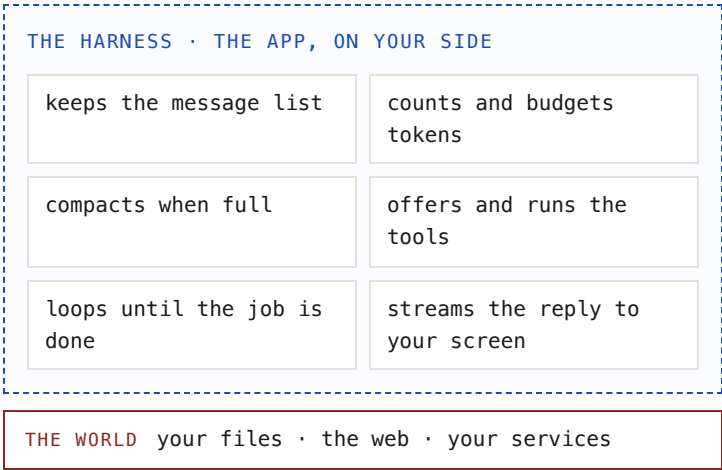
× a typical wrong answer. The true value is 31,650,328.
That exact sum appears nowhere in its training data, so it predicts digit by digit: right length, plausible shape, wrong number.

THE FIX IS THE SAME AS ALWAYS

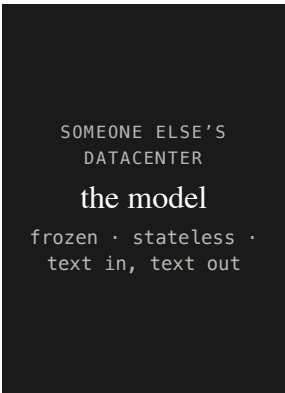


The rule of thumb: it's strong at language-shaped tasks it has seen patterns of, and unreliable at anything needing exact computation or live facts — which is precisely what tools are for.

THE HARNESS – THE SOFTWARE WRAPPED AROUND THE MODEL



TEXT
TOKENS



× not from here

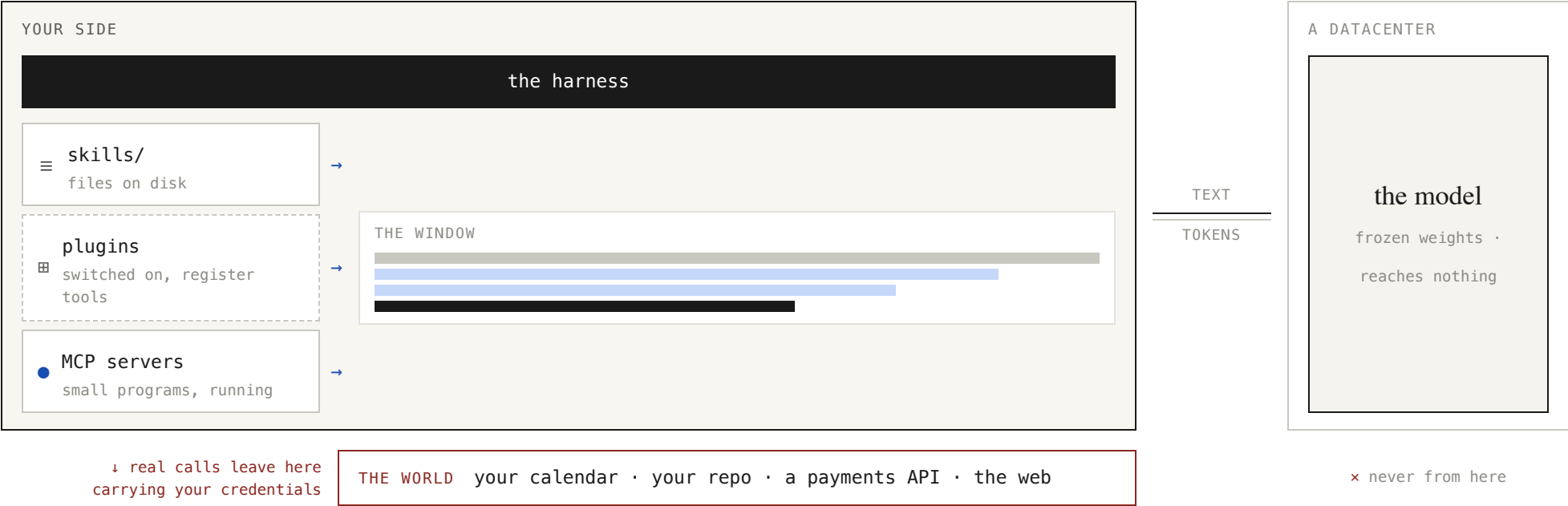
WHY THE WORD MATTERS

Almost everything you think of as “the AI” is the harness, not the model: ChatGPT, Claude, Copilot and Codex are harnesses.

Same model, different harness → very different experience. It explains why one product can browse and another can't, why one remembers your files, and why “the AI can't do X” is usually the harness, not the model.

It's also who does the forgetting: trimming, compacting, and choosing what to send are all harness decisions.

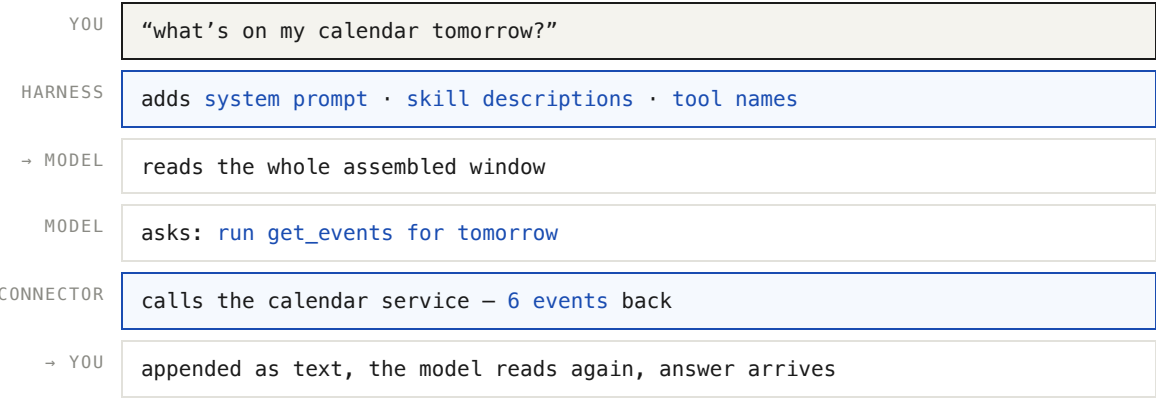
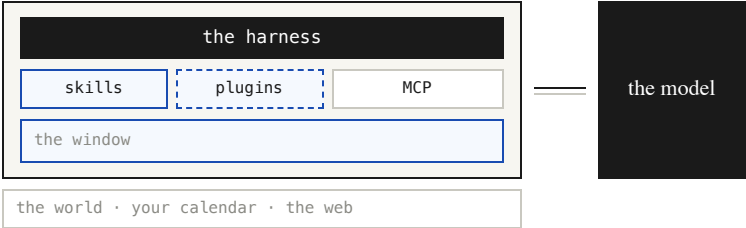
THREE PLACES – AND ALMOST ALL OF IT IS YOURS



Three different kinds of thing — a file, a package, a running program — all doing one job: getting a say in the message before it leaves your machine.

YOUR MESSAGE STOPS HERE FIRST

THE SAME MAP, LIGHTING UP



The model never touched your calendar. Every real thing that happened — the network call, the credentials, the six events — happened in the software around it.

TOOL CALLS – WHEN THE MODEL NEEDS THE OUTSIDE WORLD

ON YOUR COMPUTER

YOU

“What's today's date?”

sent across →

YOUR COMPUTER RUNS THE TOOL · THE MODEL JUST WAITS
reads your system clock → 2026-08-23, 14:02 local
appended to the list as a message · whole conversation
resent →

IN THE DATACENTER · THE MODEL

MODEL · HAS NO CLOCK AND NO CALENDAR, SO IT ASKS
← “please run `get_current_time()` and send me the
result”

MODEL · NOW WITH REAL DATA
← “Today is Sunday, 23 August 2026.”

the date is the clearest case – nothing in training could contain today. Same shape for your files, live prices, your calendar.

THREE WORDS PEOPLE MIX UP – TOOLS, SKILLS, PLUGINS

A CAPABILITY Tool · tool call A tool is something the harness can do – read a file, search, run a sum. A tool call is the model asking for one and getting the result back as text. gives it new abilities	INSTRUCTIONS Skill A written playbook for a kind of work – “here's how we build a spreadsheet”. Just text, pulled into the window when it's relevant. gives it know-how, no new powers	PLUMBING Plugin · connector A package you switch on that hands the harness a bundle of new tools – connect your email, and “search inbox” and “send draft” appear. delivers more tools
---	---	---

Skills teach · plugins add · tool calls do. And none of them change the model itself – it's still the same frozen weights reading text. They change what's in the window and what the harness can run.

Products name these differently – “actions”, “connectors”, “extensions”, “MCP servers”. Ask which of the three it is and the confusion clears.

A SKILL IS A TEXT FILE · NOTHING MORE

monthly-report.md

Monthly report

Always pull figures from the finance sheet, never from chat.

Steps

- 1. Check last month's file for the format
- 2. Fill the four sections in order
- 3. Flag anything down more than 5%

Done when

Every number has a source noted beside it.

That's the whole artefact – a plain markdown file someone wrote once. No code, no training, no new abilities. When the work matches, the harness pastes it into the window as instructions, and behaviour changes because the instructions changed.

WHY BOTHER

It saves you re-explaining your own process every time – and it means two people get the same result out of the same tool.

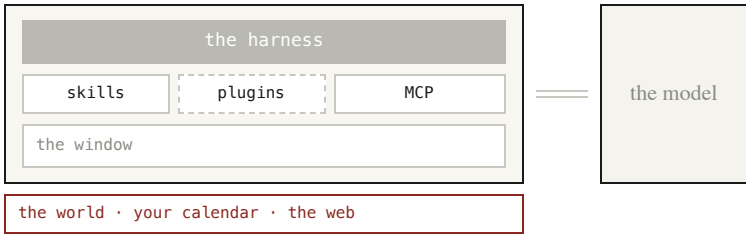
AND YOU DON'T HAVE TO WRITE IT YOURSELF

Describe the process in chat and ask for a skill file. Most tools have a skill *for making skills* – you talk, it writes the markdown, you correct it.

If you can write a good handover note for a colleague, you can write a skill. It's the same document.

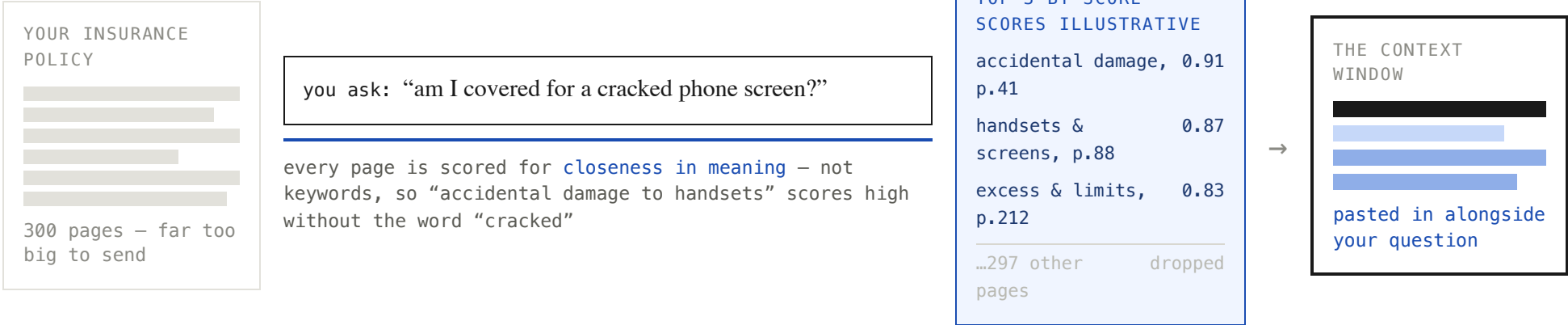
WHO ACTUALLY DOES THE WORK – ONE PLUGIN, STEP BY STEP

THE SAME MAP, LIGHTING UP



- YOUR SIDE
 - you connect the calendar plugin. Your side keeps the login and tells the model only that a tool called `list_events` exists.
- THE MODEL
 - all it can emit is a line of text asking for that tool. It cannot call anything.
- YOUR SIDE
 - signs in, calls the service, retries on failure, cuts 200 events down to tomorrow's six.
- THE MODEL
 - those six lines go back as a message, read like any other text.
- SO...
 - it never held your password or opened a connection. When a plugin misbehaves it is the plumbing, not the model.

YOUR OWN DOCUMENTS – THE POINT: YOUR FILES NEVER GO INTO THE MODEL



Nothing was “trained on your data”. Three pages were pasted into the window, read once, and forgotten — the model is answering from text you handed it a second ago.

Which explains the failure you'll actually hit: if the search picks the wrong pages, the answer is wrong – and the model has no idea it's missing anything.

WHAT IS AN “AGENT”, EXACTLY?

A PLAIN CHATBOT REPLY

you ask

 →

it answers

 ■

one pass. Whatever it knew, you get. No looking, no checking.

AN AGENT · SAME MODEL, RUN IN A LOOP

UNDERSTAND

 “fix the broken link on the pricing page”

LOOK

 reads the page, finds the link, checks where it points

ACT

 calls a tool to edit the file

CHECK

 loads the page again to confirm it works

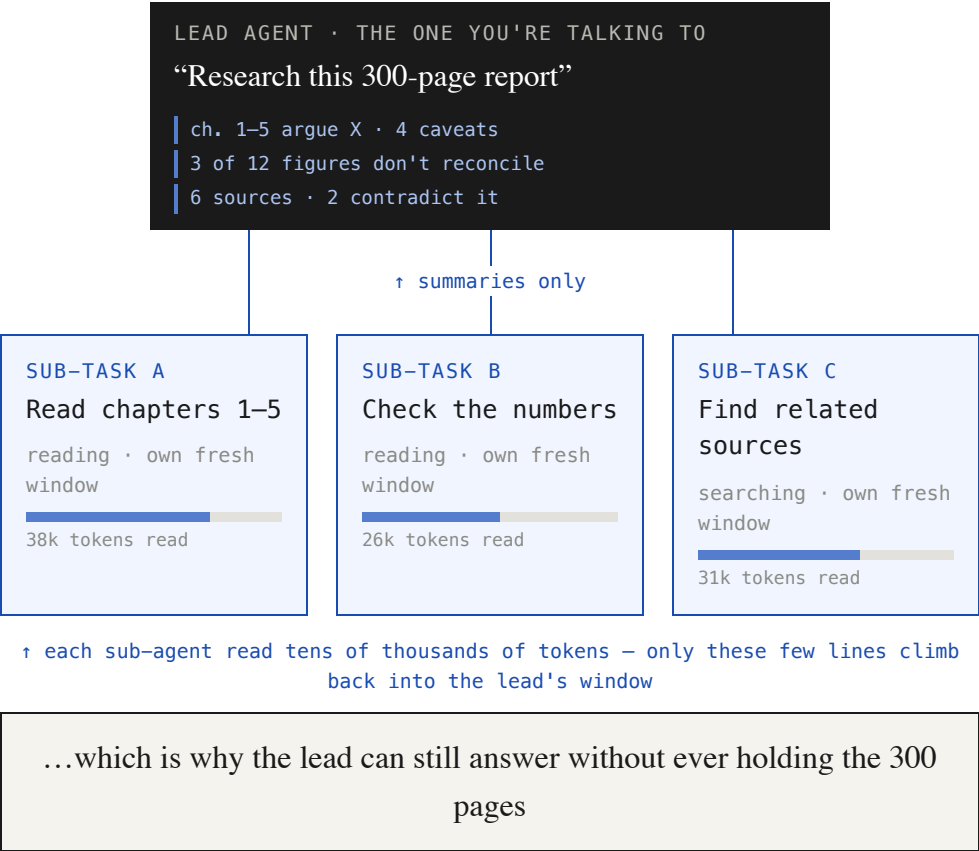
↺ still broken? back to “look” and round again – that repetition is the whole difference

“ACTING” = CALLING A TOOL read a file · run the tests · search the docs · click through a web app in a real browser · take a screenshot · call an API – each result comes back as text in the window

You talk to *one* agent per conversation. It may hand bounded pieces of work to helper agents — but they're workers, not colleagues you have to manage.

The one you're talking to stays accountable for the final answer. It's not “one agent per task” – it's one agent per request, plus however many helpers that request needs.

SUB-AGENTS – SPLITTING A JOB TOO BIG FOR ONE CONTEXT WINDOW



THE KEY IDEA

Three fresh windows instead of one overflowing one. The lead holds only the summaries — that's the real reason to split work, not magic.

Worth it when the pieces are genuinely independent. For small, tightly connected work one agent is clearer – and the lead stays accountable either way.

PUTTING IT TOGETHER – WATCH ONE JOB GO ROUND THE LOOP

YOUR COMPUTER

holds the list · runs tools

answer shown; you reply again

← the reply comes back

THE MESSAGE LIST340 TOKENS

system prompt

you: "what's today's date?"

model: run get_current_time()

tool result: 2026-08-23

model: "Today is 23 Aug 2026."

you: "and the day after?"

it only ever gets longer – every trip resends all of it

THE MODEL

reads the list · replies once

idle – a fresh read next trip

trip 3 of the loop An “AI agent” is only this: the list grows, it gets resent, the model answers or asks for a tool. The loop lives on your side.

THE THREE THINGS YOU ACTUALLY CONTROL

1 · Your prompt

everything it knows about the job arrives this way

the goal – what you want

the constraints – what it must respect

what “done” looks like

leave one out and it will cheerfully invent it

2 · Skills

reusable instruction packs, written once

“how we build a spreadsheet”
a playbook – steps, conventions,
sometimes scripts

it gets pulled into the window when the work matches, so you don't re-explain the same process every time. It's instructions – not a smarter model.

3 · Which model

and you can switch part-way

fast + cheap → sorting, drafting, bulk edits

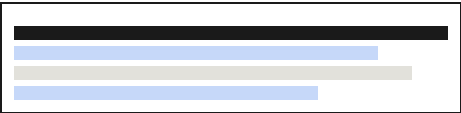
strong reasoning → the tricky bug, the design call

the catch: a new model reads the same window differently – so keep the plan and decisions written down, not implied.

Notice what all three have in common: they're just text you put in the window. That's the whole lever — there's nothing else you can reach.

RECAP – THE WHOLE THING IN FOUR IDEAS

The list



A chat is a list of messages. It's resent in full every turn – that's the only reason it seems to remember.

The box



The window is a fixed box measured in tokens. Only what's inside exists; the overflow is genuinely gone.

The guess



It only predicts the next token. Fluent and confident even when wrong – and useless at exact sums.

The loop



Everything else – tools, skills, agents, sub-agents – is software running that loop around the box.

A frozen text predictor, handed a list of messages in a fixed box, run in a loop by ordinary software that does all the fetching.

Hold onto that and the rest is detail: hallucinations are the predictor, forgetting is the box, and anything it “does” is the loop calling a tool.