

Getting started with AI agents

Chat & code — what they are, how to drive them, and what to watch out for.

*“Why, Mr. Anderson? Why do you persist... in writing it all by hand?”
— Agent Smith*



Ways in

Many brains, several cockpits

one API to route them all



Chat apps

Claude, ChatGPT, Qwen Chat, Z.ai — in the browser, desktop or phone. Nothing to install.

Many now take on longer tasks and hand back finished documents and spreadsheets.



Coding agents

Claude Code, OpenAI Codex, opencode — in a terminal or a desktop app.

They read your repo, edit files and run commands — with your approval.



Plugins

The assistant inside tools you already use:

VS Code and JetBrains IDEs, the browser, Office and Slack. Most vendors ship them.

Gateways

OpenRouter or a LiteLLM proxy put many models behind one API — and an open agent like opencode can drive any of them.

What it's good for

Where it earns its keep in LLRF

01

Read & refactor code

Explain, tidy or port C++, Python or VHDL — controller firmware, FPGA logic, code nobody remembers writing.

02

Write quick scripts

SCPI / PyVISA instrument control, loop-data crunching, transfer functions and Bode plots.

03

Dig through logs

Trips and interlocks: patterns a human would miss — "something happens every 3 min 59 s".

04

Review & tests

Cheap, patient code review. Drafts of unit tests and HDL test benches.

05

Docs & email

Summarise manuals and datasheets; draft procedures, abstracts and messages.

06

Grind work

Repetitive migrations across many files or projects — new OS, new framework version.

↑ *finally, someone who reads the datasheet*

The model

Why the model makes things up

01

Trained, then frozen

It learned from lots of text up to a cutoff date. On its own it looks nothing up.

02

Predicts, doesn't recall

It writes the most plausible next piece. Usually right — but plausible is not the same as true.

Catch it: Ask for sources and open them → run the code → recompute the numbers → search the web for anything current.

*the answer is 42. the citation, unfortunately,
is invented.*

Spot the fake

```
rm = pyvisa.ResourceManager()
```

✓ real

```
pm.query_power_dbm(channel=2)
```

⚠ invented

Smith et al., PRAB 27, 2024

🔍 check it

A hallucination looks **exactly** like a fact.

Context

Context is short-term memory

Everything the model can see lives in one window, measured in tokens ($\approx \frac{3}{4}$ of a word). Many models hold up to a million — and all of this shares it:



It all counts

Every step re-reads the whole conversation, and every token is billed.

It fills up

Near full, quality drops and it compacts: older detail is summarised and blurs.

One task per session

/clear or start fresh. No emails in your VHDL-debugging session.

Caching, briefly

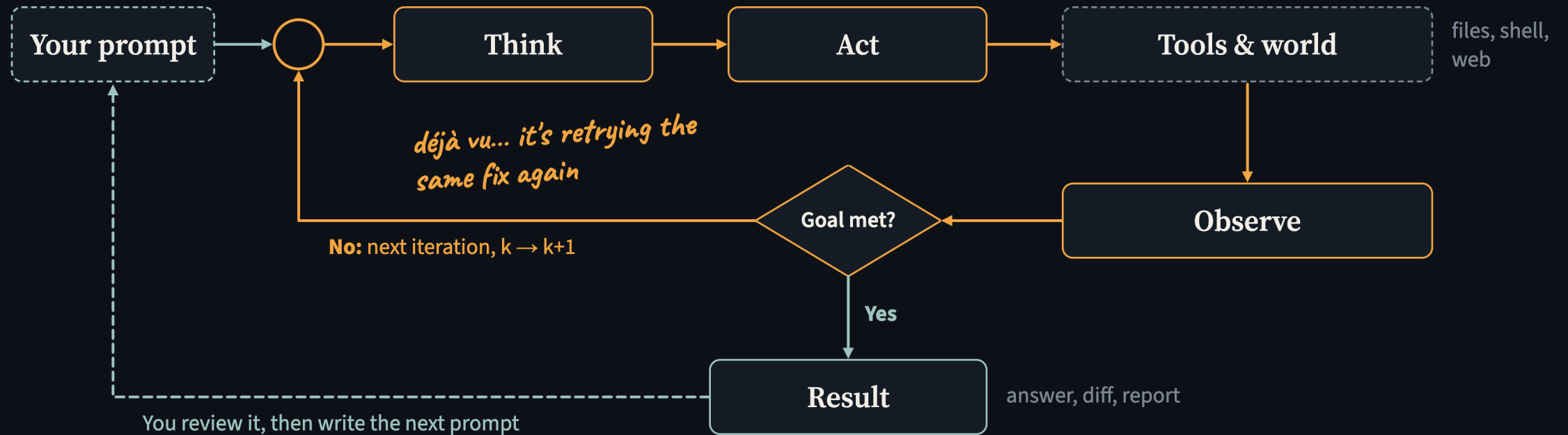
Repeated context is re-read cheaply — but the cache expires within minutes.

all those tokens... lost in compaction, like tears in rain

Two loops: the agent's and yours

the operator is always the outer loop

The model decides each step; the harness runs the loop until the goal is met. Then the result comes back to you, and your next prompt starts it again.



The line-up

Pick the lightest model that works

Small and fast (\$) → everyday (\$\$) → frontier (\$\$\$\$)

Anthropic	Haiku 5.5 · Sonnet 5.5 · Opus 5.5 · Fable 5.1	small → frontier
OpenAI	GPT-5.6 Sol · Terra · Luna	GPT-6 Astra: limited release
Z.ai	GLM 5.3	open weights · 1M context
Alibaba	Qwen3.8 family — Flash to Max	open weights

Start with an everyday model; move up when it genuinely struggles. Bigger models and higher effort burn more tokens — you pay per token, in and out.

versions valid until roughly the coffee break

Tools

Tools reach beyond text

The model only writes text. A tool lets that text do something — and brings fresh, real information back.

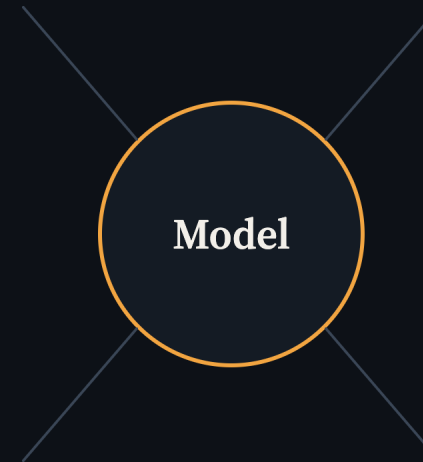
Why MCP: the connector holds the credentials and does the work; the model sees only the actions you expose — read-only, if you like. Even specific equipment can sit behind one.

The catch: a tool is real access to real systems and real data. That's why permission matters.

so preoccupied with whether it could run the command, nobody asked if it should

Web search
current facts

Files & code
read · edit · run



Shell
scripts · builds

Connectors
MCP · your systems

Big files: let it fetch, don't paste

- | | | |
|---|------------------------|--|
| 1 | Manuals & datasheets | Have it convert PDFs to text and build an index. Next session it reads the index first — your library grows as you go. |
| 2 | Put files, don't paste | Drop them in the working folder and say what you need. It reads only the relevant parts — a 35 MB PDF won't fit in the context anyway. |
| 3 | Data, not pictures | Give it the original data at full resolution (CSV, HDF5), not plots. Images are quantised — a spike may never have been drawn. |

Screenshots and schematics: **for understanding, not for numbers.**

we're gonna need a bigger context window

Skills

Skills teach it your way of working

*"I know VHDL."
"Show me."*

A small folder of instructions the agent picks up when a task matches — so you stop re-explaining your process.

- › **Often built in:** many agents ship skills for Word, Excel and PDF files.
- › **Cheap on context:** only the short description is always loaded.
- › **Easy to make:** after a task goes well, ask the agent to turn it into a skill.
- › **Portable:** an open, plain-text format read by a growing list of agents.

*write it once, stop explaining it
every Monday*

skills/test-report/SKILL.md

```
---  
name: test-report  
description: Build our  
standard test report. Use  
when asked for one.  
---  
  
# Steps  
1. Pull readings from the log  
2. Compute the key figures  
3. Fill the report template
```

Working with it

Run it like a project

1

One task, one working directory, one session.

2

Sources in one place, outputs in another.

3

Keep it in Git — and say when to commit and push.

4

Write standing rules once; they load every session.

| **Set clear objectives.** Say what "done" looks like, use direct wording ("do X", not "we need to"), and review the result before you rely on it.

don't cross the streams: one task per session

Permission

You are the gate

Before it edits a file, runs a command or reaches a system, it asks.

That prompt is a decision, not a formality.

"I'm sorry, Dave. I'm afraid I need approval for that."

Auto-approval lets clearly safe, routine steps through and still stops on risky ones. Fewer prompts — the responsibility stays with you. When unsure, deny and ask it to explain.



Approve once

Read what it's about to do. Fine for this step? Allow it — and watch.



"Always allow"

You hand over the wheel. Only for safe, reversible actions.



Least privilege

Trust only the project folder. Nothing irreversible on trust.

Know where your data goes

prompt + files + tool results —————▶ the vendor's servers



What happens

Unless you self-host, everything you send is processed remotely — the command line too. Whether it's kept or trained on depends on vendor, plan and settings: check them.



What to think about

Personal data, yours and colleagues'. Unpublished work. Anything you may patent. NDA material. Passwords and keys. Your lab's own AI policy.



Ways to reduce risk

Anonymise before you paste. Share the parameters, not the device name. Keep sensitive folders out of reach — an agent will find them. Or run open weights in-house.

You decide what to share — **just decide knowingly.**

your data is out there.

When data must stay in-house

Why

Nothing leaves your machine or your lab.
You choose the model, the version and when to upgrade.

What you need

A runner such as Ollama or vLLM, ideally behind a LiteLLM proxy so any agent sees one API. A quantised ~30B model fits a laptop with 48–96 GB of RAM.

Limits

Laptop-size models are well behind the frontier. The big open ones, like GLM 5.3 or Qwen3.8-Max, need a serious GPU server.

Try: Qwen · GLM · Gemma · Granite · Mistral

Open-weight families on Hugging Face — check the current versions and licences.

runs on a laptop. also heats the laptop.

Limits

It can be wrong

- ⚠ **Exact numbers.** It's not a calculator — have it write Python that does the maths.
- ⚠ **Current or niche facts.** Training has a cutoff, and it can be thin on specialised topics.
- ⚠ **What it can't see.** Live hardware, private systems, data you haven't given it.
- ⚠ **Equipment without safeguards.** It can dream up a higher setpoint.

Accountability

— and it's on you.

It works under your name and your login. If it trips an interlock because you asked for more voltage, that's your trip.

Read the code. **Check** the numbers. **Test** the change.

Fluent output is not verified output.

"The AI said so" is never a defence.

*"LLMs are like any other machine. they're either a benefit or a hazard. if they're a hazard, it's your problem."
— Deckard, more or less*

Prompting well

Improve the signal-to-noise ratio

Vague



"fix my script"

→ It guesses, asks a lot back, or fixes the wrong thing.

Specific



"This Python reads an RF power meter over PyVISA but times out. Here's the code and the error. Fix the timeout, keep the SCPI protocol, and explain the change so I learn."

→ Does what you meant, usually first try.

↑ *garbage in, eloquent garbage out*

Goal + context + constraints + the format you want back

Bigger jobs

Plan first, then execute



From a real session

SPI state machines scattered over a VHDL repo → discovery report → plan to consolidate on a RISC-V core → plan revised to build test benches first.

Don't accept a plan too early — once accepted, it starts executing.

premature execution is the root of all evil

Things worth saying

- › Scan the repo, find the ... and write a report.
- › Plan the ... and write it to plan.md.
- › Explain the change so I learn.
- › Turn what we just did into a skill.

“explain so I learn” – or you’re the one being trained

- › Save this as summary.md with a table.
- › Add test benches; run them before and after.
- › Write a report of what you did.
- › Install this skill from <URL>.

start fresh between tasks

check your spend

repeated job → script or skill

one-off → just let it look

Go deeper

References & links

questions? ask a human. or the agent, but check it.

Chat & models

claude.ai — Anthropic

chatgpt.com — OpenAI

chat.qwen.ai — Alibaba Qwen

z.ai — Z.ai GLM

gemini.google.com — Google

mistral.ai — Mistral

Agents & gateways

docs.claude.com — Claude Code

openai.com/codex — Codex

opencode.ai — open source, any model

openrouter.ai — many models, one API

docs.litellm.ai — self-hosted gateway

Open weights

huggingface.co — the weights

ollama.com — local runner

A poor answer? Use the feedback buttons — vendors read them.

Skills and MCP connectors are open formats — what you learn carries over between tools. Workshop: indico.elettra.eu/event/63