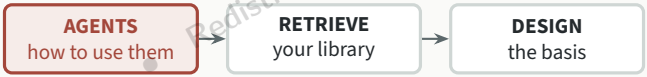
Session 6.1

Agentic AI: How I Use It, What It Is, How It Works, and How to Use It Well

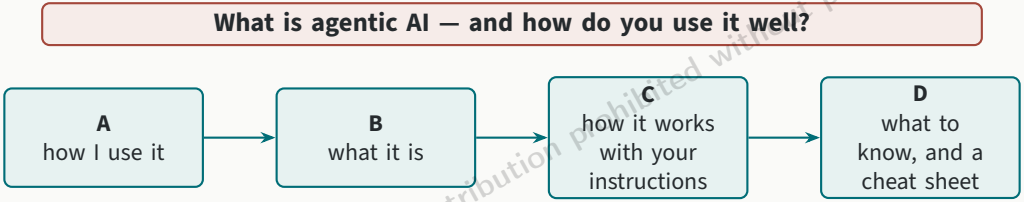


Redistribution prohibited without permission

Zhigang Feng

© Zhigang Feng

The Question, and the Route



Redistribution prohibited without permission

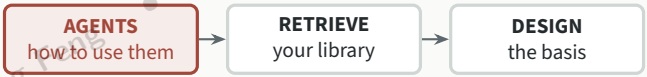
© Zhigang Feng

- Session 5 ended with a model that gives advice. This one acts: it runs code, moves files, drafts.
- Four parts: how I use it, what it is, how it works with your instructions, what to know to use it well.
- **You stay the project manager. The agent does the typing.**

Session 6.1

How I Use It

An overview: a cluster, a library, one method



© Zhigang Feng Redistribution prohibited without permission

How I Work: Everything on a Cluster, and a Library the Agent Can Read

NSF clusters: 1,600 CPUs, 20+ H100 GPUs

the agent, at the project root:
code, data, jobs, drafts, slides

compute
solve and train the models of Sessions 2–4

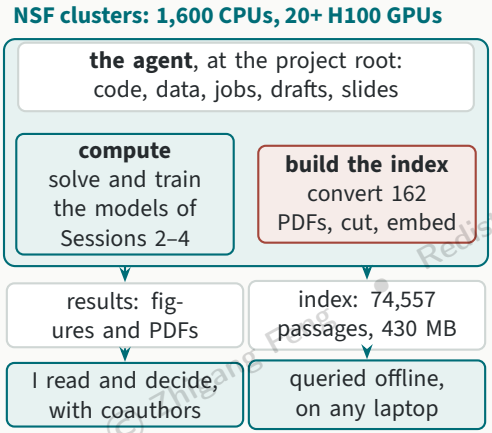
build the index
convert 162 PDFs, cut, embed

results: figures and PDFs

index: 74,557 passages, 430 MB

I read and decide, with coauthors

queried offline, on any laptop



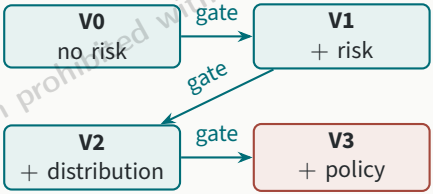
- The agent runs where the code, data and jobs are: a login node. What leaves the cluster is a PDF I read.
- It also answers from my own library, citing the page, or says the evidence is weak: **RAG**, Deck 6.2.
- **Build where the data and the GPUs are; read and decide anywhere.**

Source: The instructor's retrieval system and project record, read 12 October 2026; allocations through NSF grants. Deck 6.2 builds the index.

© Zhigang Feng
Redistribution prohibited without permission

One Method: Solve the Easy Version First

- Start from a version you understand, with a benchmark you trust.
- Add one margin at a time; a new bug belongs to the margin you just added.
- **Never ask the agent for the full problem in one jump.**



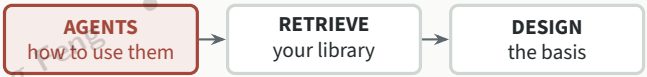
homotopy: deform a solved problem toward the target, one validated step at a time; a failed step returns to the last version that passed

© Zhigang Feng
Redistribution prohibited without permission

Session 6.1

What It Is

Chatbot, copilot, agent, team — and what makes AI agentic



© Zhigang Feng Redistribution prohibited without permission

From a Chat Box to Agent Teams, 2022–2026



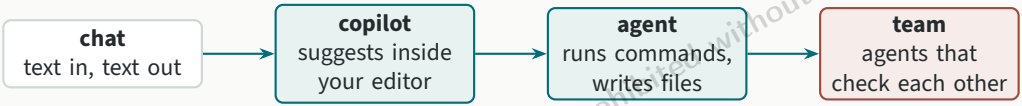
2022	2023	2024	2025	2026
answers a chat window reaches the public	**calls tools** structured tool calls in the APIs	**shares plumbing** one protocol for tools and data	**acts on your files** terminal agents, sub-agents	**runs unattended** scheduled and cloud runs
ChatGPT (Nov 2022)	function calling (Jun 2023)	one tool protocol (Nov 2024)	it goes multi-vendor (Mar 2025)	scheduled runs (Apr–May 2026)

- Each year the model could touch more of your machine: answer, call tools, share plumbing, act on files, run unattended.
- **Each step made the work cheaper. None made checking it cheaper.**

Source: Dates checked against vendor announcements, September 2026.

© Zhigang Feng
Redistribution prohibited without permission

Chat, Copilot, Agent, Team: A Spectrum, Not a Switch



- What changes along the strip is not the model. It is how much of your machine it may touch.
- One measured point: 79% of terminal-agent conversations were automation, against 49% in a chat window.
- **Choose a position per task, and move left when the task is delicate.**

Source: Anthropic Economic Index, “AI’s impact on software development”, 28 Apr 2025 (read 23 Sep 2026).

© Zhigang Feng Redistribution prohibited without permission

What Makes AI Agentic: Four Marks, All Needed

tool use
calls Python, R, Stata, the shell

autonomy
a goal, not a prompt: it plans the steps

iteration
reads the error, fixes, reruns

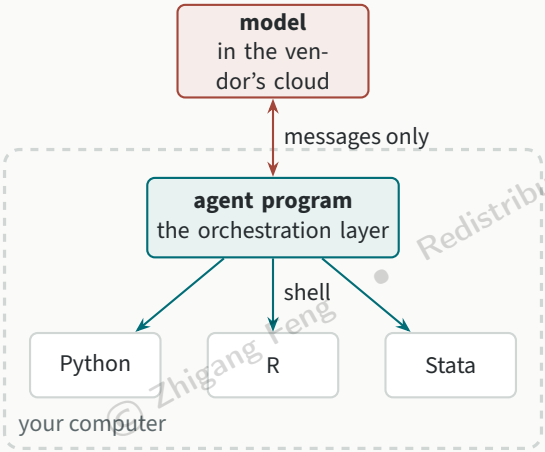
persistence
conventions and plans kept on disk

An **agent** is a model in a loop, given tools and pointed at a goal: the cleverness is in the model, the *capability* in the loop.

- The marks stack: tools without autonomy is a console; autonomy without iteration dies at the first error; nothing survives the session without persistence.
- **"Agentic" is a degree: how much of the loop runs without you.**

© Zhigang Feng
Redistribution prohibited without permission

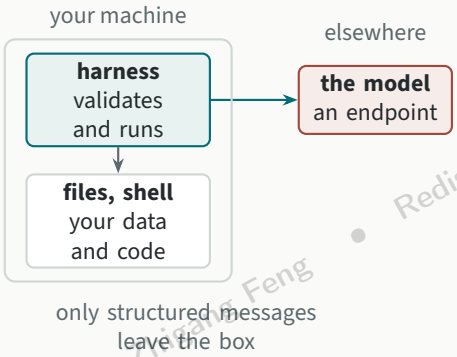
How an Agent Works on Your Machine



- The agent program runs on your machine and does no econometrics itself.
- It asks the model "what next?", then runs that step in the Python, R or Stata you already have.
- **You keep Stata. You gain a layer that types commands, reads errors, and tries again.**

© Zhigang Feng Redistribution prohibited without permission

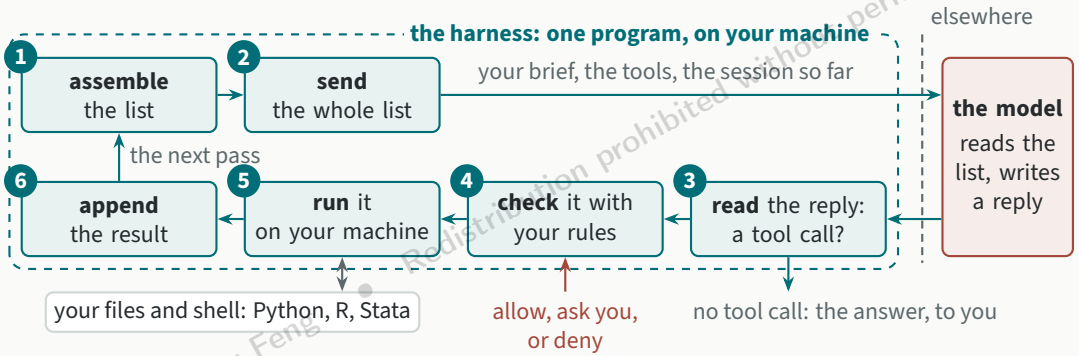
Three Parts, and Only One of Them Is Clever



- **The harness** is an ordinary program on your computer: it runs commands and enforces the rules you set.
- **The model** is an endpoint elsewhere: it reads messages and names the next call. That is the whole of it.
- **The intelligence sits in one part; the discipline in the other two.**

© Zhaoyang Feng
Redistribution prohibited without permission

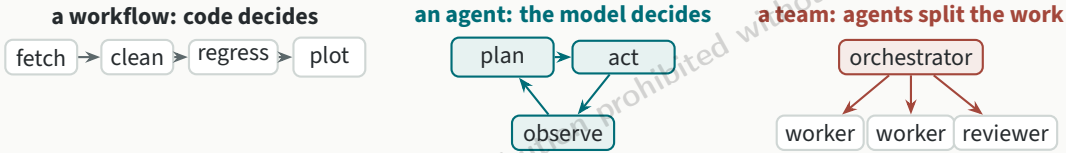
How the Harness Works: Six Steps, Repeated Until No Tool Is Called



© Zhigang Feng Redistribution prohibited without permission

- The model only writes; every read, check and run on your machine is the harness's.
- A refused call comes back as text, like any output; the next pass sends the longer list.
- **Your levers all sit in the harness: the brief, the tools, the rules.**

Agent, Workflow, Team: Who Decides the Next Step



Unauthorized redistribution prohibited without permission

- Workflow: code decides the steps. Agent: the model decides. Team: agents split the work and check each other.
- Cheap and predictable; handles surprises; buys breadth and pays in coordination. Start at the left.
- **Capability and risk both sit with whoever chooses the next step.**

© Zhiqiang Feng

Where the Agent's Files Live: Your Home Folder and Your Project Folder

your home: every project you open

`~/.claude/`	read for every project
├─ `CLAUDE.md`	your standing brief
├─ `settings.json`	your rules and hooks
├─ `skills/ agents/ rules/`	yours, everywhere
├─ `plans/`	plan-mode drafts
└─ `projects/<project>/`	
├─ `memory/MEMORY.md`	auto memory
└─ `<session>.jsonl`	session transcripts
`~/.claude.json`	servers and sign-in

the project: this one, shared with the team

`my-project/`	where you start
├─ `CLAUDE.md`	the brief, committed
├─ `CLAUDE.local.md`	private, gitignored
├─ `.mcp.json`	tool servers, shared
├─ `.claude/`	
│ ├─ `settings.json`	team rules, committed
│ ├─ `settings.local.json`	yours, gitignored
│ ├─ `rules/*.md`	rules by topic
│ └─ `skills/ agents/`	skills, sub-agents
└─ `src/ data/ paper/`	the work itself

- Same layout, two homes: one for every project you open, one for this project.
- Committed files travel with the repository; `.local` files stay out, on your machine.
- **Both load at every start: briefs add up; settings rank, project above yours.**

Source: Claude Code 2.1.303: documentation on memory, settings, skills and MCP, read 20 October 2026.

© Zhiyang Feng • Redistribution prohibited without permission

Session 6.1

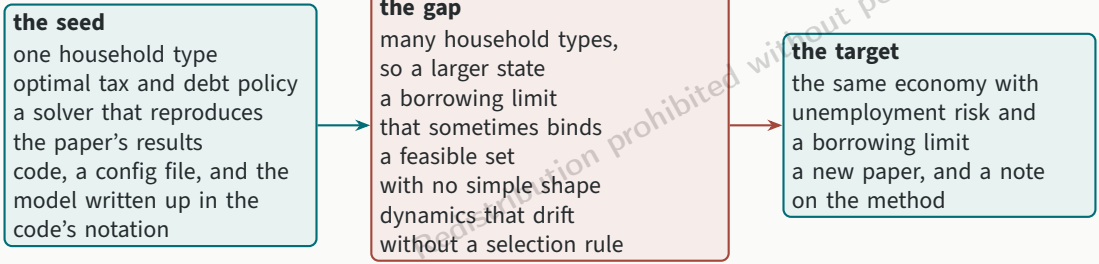
Your Instructions, the Loop, the Workflow

You are the project manager: one project from seed to paper, and how the agent runs each step



© Zhigang Feng Redistribution prohibited without permission

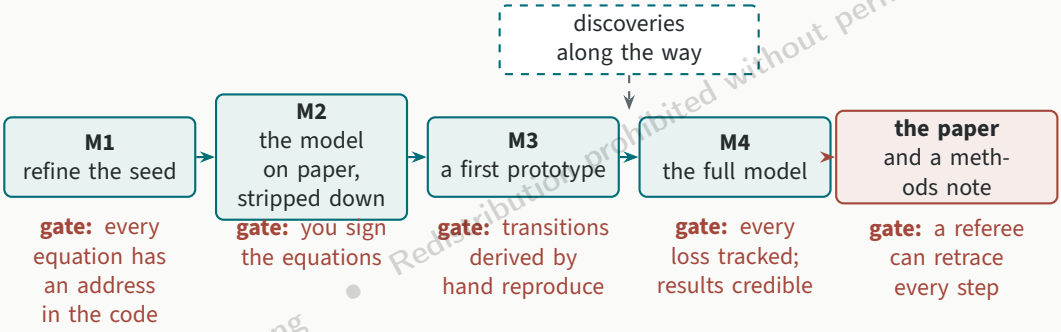
The Project: A Working Seed, a Target, and the Gap Between Them



© Zhigang Feng Reproduction prohibited without permission

- A good seed: it works, it is documented, it is modular, and it points at the target.
- Each item the seed lacks becomes one milestone; two of them became new methods.
- **The plan is the list of what the seed lacks.**

The Plan: Four Milestones, Each With Its Gate



© Zhiyang Feng Redistribution prohibited without permission

- One margin at a time: a new bug belongs to the margin you just added.
- Gates are written first; a failed step returns to the last milestone that passed.
- **A milestone is a validated artefact on disk, not a feeling of progress.**

How the Agent Runs a Step: Your Instruction, a Tool Call in JSON, an Observation

you	Restate the model and the algorithm in your words. Then audit the code against the text.
model →	`{"tool": "read_file", "path": "model.tex"}`
harness ←	the write-up, as text
model →	`{"tool": "search", "pattern": "euler", "in": "src/"}`
harness ←	three matches, with file and line
model →	`{"tool": "shell", "cmd": "rm -r old_runs"}`
harness ←	refused: not on your allow-list
model	no tool call: (7) is in model.py, (9) is not implemented, the score formula differs.

© Zhaang Huang Redistribution prohibited without permission

- A tool is a typed function you wrote: a name, a schema for its arguments, a description.
- The model answers with one JSON line; the harness runs it and sends the output back.
- **It stops when the model stops calling tools. Your rules can refuse any call.**

Step 2: The New Model on Paper, Before Any Code

you Add unemployment risk and a borrowing limit; write the new equations. No code yet.

model → `{"tool": "write_file", "path": "model_v2.tex", "text": "..."}`

harness ← written

you read the draft. Two plausible equations are wrong:
a budget with next period's consumption; a transition probability running the wrong way.

you Fix both. Keep utility simple and the risk symmetric.

model → `{"tool": "edit_file", "path": "model_v2.tex", "old": "...", "new": "..."}`

harness ← written

- The draft compiles and reads well; no compiler catches an economic error. You do.
- Simplicity is a design decision: log utility and symmetric risk, chosen to be checkable.
- **Gate: the equations carry your signature before a line of code exists.**

© Zhiang Yang
Redistribution prohibited without permission

Step 3: A First Prototype — Write, Run, Read the Error, Fix

you	Implement model_v2.tex in the seed's module layout. Keep the config file.
model →	`{"tool": "write_file", "path": "src/model.py", "text": "..."}`
model →	`{"tool": "run", "cmd": "python train.py --config config.json"}`
harness ←	after a dozen iterations the loss is NaN
model →	`{"tool": "edit_file", "path": "src/model.py", "new": "clamp c_next"}`
model →	`{"tool": "run", "cmd": "python train.py --config config.json"}`
harness ←	residuals falling; the feasible region is drawn
model →	`{"tool": "run", "cmd": "pytest tests/transitions.py"}`
harness ←	all passed: the forward pass matches the hand-derived transitions

© Zhiqiang Feng ● Redistribution prohibited without permission

- An error message is not a failure; it is the next observation. Read, edit, rerun.
- Same module skeleton as the seed, so every diff stays readable to you.
- **Gate: the forward pass reproduces hand-derived transitions, kept as a test.**

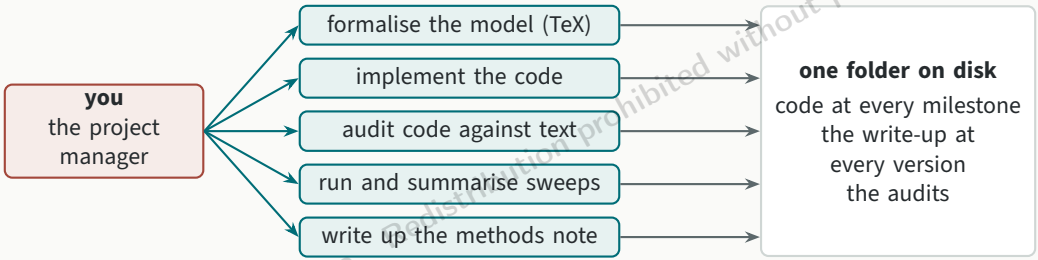
Step 4: The Full Model — Failures Become Structure, Agents Run the Sweeps

you	Paths escape the feasible set; the critic oscillates. Diagnose and propose structure.
model →	`{"tool": "write_file", "path": "PLAN.md", "text": "hypotheses, next runs"}`
model →	`{"tool": "spawn_agent", "task": "run the sweep over config.json; report one table"}`
harness ←	the table, one row per configuration
model →	`{"tool": "spawn_agent", "task": "audit src/ against model_v2.tex"}`
harness ←	two mismatches, with file and line
model	no tool call: project paths back; freeze a target critic; track each loss apart.

- A sub-agent is a second loop with its own window: one runs the sweep, another audits.
- The plan file is memory: it outlives the window and the session.
- **Gate: every loss tracked apart, and a result you find credible.**

© Thiago S. Ferro · Redistribution prohibited without permission

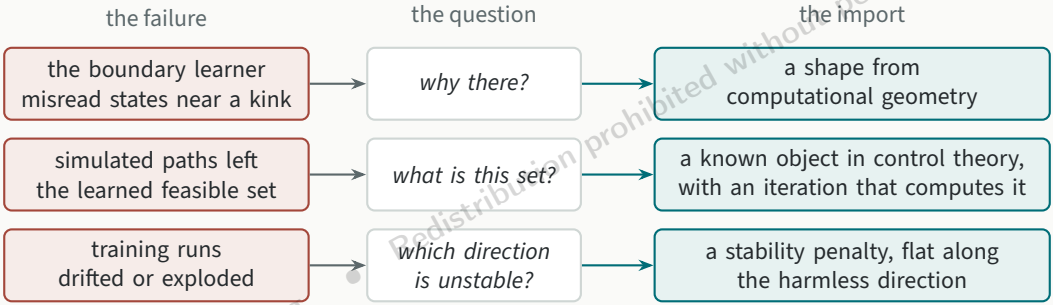
Orchestration: One Manager, Several Sessions, One Folder



© Zhiyang Feng Redistribution prohibited without permission

- One session per layer, each a fresh window; the folder on disk is what they share.
- The reviewer is never the builder: a second session audits what the first one wrote.
- **You integrate. No agent sees the whole project; you do.**

The Creative Part: Three Failures That Became New Methods



the failure	the question	the import
the boundary learner misread states near a kink	*why there?*	a shape from computational geometry
simulated paths left the learned feasible set	*what is this set?*	a known object in control theory, with an iteration that computes it
training runs drifted or exploded	*which direction is unstable?*	a stability penalty, flat along the harmless direction

- Ask why, not for a fix: the agent diagnoses and searches, you name the economics.
- Each import came from a literature economists rarely read, in days rather than years.
- **Three parts of the final algorithm were not in the plan.**

© Zhigang Feng Redistribution prohibited without permission

What the Manager Provides, What the Agent Provides

you provide	the agent provides
the seed and the destination	speed between them
economic correctness	syntactic and structural correctness
the gate at each milestone	candidate artefacts to accept or reject
the question “is this credible?”	the diagnostics to answer it

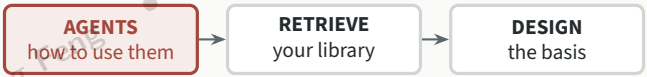
© Zhigang Feng · Redistribution prohibited without permission

- Four things: a working seed, a destination, judgement at the gates, orchestration.
- The agent chose no question, caught no timing error, judged no result credible.
- **The job is the cycle, and the cycle is management.**

Session 6.1

Inside the Machine

What to know to use it well, and one page to take home



© Zhigang Feng Redistribution prohibited without permission

The Session Is a List, Not a Connection

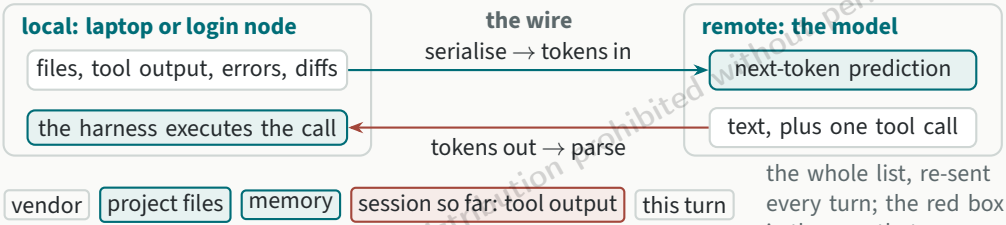


teal: the only new bytes;
white: re-sent word for word,
and paid for again.

- The endpoint keeps nothing. Each turn re-sends the whole conversation so far.
- "The window is full" means the list outgrew the limit, and it grows fastest from tool output, not from what you type.
- **Clearing, compacting and resuming are all operations on a list.**

© Zhigang Peng
Redistribution prohibited without permission

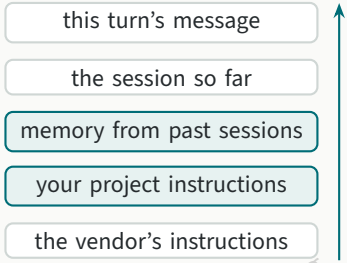
What Crosses the Wire



© Zhiyang Feng Redistribution prohibited without permission

- Nothing of yours moves and the model never logs in. What crosses is text: a tool call out, an observation back.
- Any file it reads crosses too. What it may open is an authorisation decision.
- **The meter counts both directions, and the part that grows is what your tools printed.**

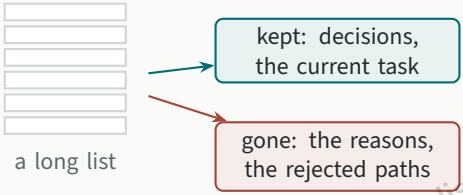
Five Layers of the Context Window



- Every turn the model decides from one place only: whatever is in this window now.
- The vendor's layer and this turn's message are not yours to manage. Your project instructions and memory are, and both are files on disk.
- **The binding constraint is attention, not size.**

© Zhaoyue Feng
Redistribution prohibited without permission

Compaction: What It Keeps, and What It Drops



- When the window fills, the harness summarises and carries on, on its own unless you do it first.
- Lossy in a specific way: conclusions survive; the reasons, and the small corrections you made in passing, often do not.
- **State belongs in files. Write down what must outlive the window, and compaction costs you nothing.**

© Zhigang Feng
Redistribution prohibited without permission

Tool, Skill, Sub-agent: Where the Reasoning Lives

tool	**skill**	**sub-agent**
code you wrote	prose you wrote	a second loop
the model picks the *arguments*	the model picks *to read it*	its own context and tools
no reasoning inside	reasoning stays in the main loop	reasoning happens elsewhere
costs a schema	costs one description line	costs a second context
fails by raising an error	fails by being ignored	fails by drifting off-task

© Lirang Feng
Redistribution prohibited without permission

- A tool is code you wrote: the model picks the arguments. A skill is prose you wrote: the model picks to read it. A sub-agent is a second loop with its own list.
- Loading order is cost order: rules at every start, a skill's body on match, a sub-agent in a fresh list.
- **Ask of any new agent term: once installed, where does the reasoning happen?**

A Skill Is a Folder: One Key File, the Rest Read Only When Needed

the folder

```
.claude/skills/                  in the project
└─ fomc-library/                 the name you type
    ├─ SKILL.md                  the key file
    ├─ scripts/toy_rag.py        run, not read
    └─ references/cite.md        read if needed
```

when each part is read

- always in the list: name and description
- once used: the rest of SKILL.md
- on demand: a reference read, a script run

- **The folder holds a team's working knowledge, not a task.**

SKILL.md

when

```
---
name: fomc-library
description: Answers questions about FOMC
  statements from the local index, citing
  each claim as [file#n]. Use when asked
  what the Committee said at a meeting.
---
```

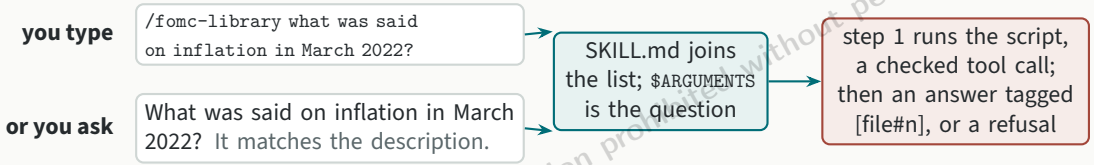
how

```
1. Run scripts/toy_rag.py on the question:
   ask index/ "$ARGUMENTS"
   (add --where fomc_YYYYMM for a meeting)
2. Answer only from the passages it prints;
   tag every claim with its [file#n].
3. If none answers, say so. How to cite:
   references/cite.md
```

Source: Claude Code 2.1.303 skills documentation, read 21 October 2026; the open format: agentskills.io.

© Zhigang Feng • Redistribution prohibited without permission

Two Ways to Start a Skill: Type Its Name, or Let the Model Match It



in the front block	you start it	the model	for
nothing (the default)	yes	yes	most skills, like this one
`disable-model-invocation: true`	yes	no	side effects: a deploy, a send
`user-invocable: false`	no	yes	background knowledge

- Unused, a skill costs its description; once used, its body stays in the list.
- **Write the description for the model: it decides from those lines alone.**

Source: Claude Code 2.1.303: skills documentation, read 21 October 2026.

© Zhigang Feng Redistribution prohibited without permission

Cheat Sheet: Commands, Keys and Modes

set up	`/init` write the brief · `/memory` edit it · `/permissions` allow, ask, deny · `/hooks` · `/skills` · `/agents` · `/mcp`
drive	`/plan` propose, change nothing · `/model` · `/effort` low to max · `/context` the window as a grid · `/compact` · `/clear` · `/resume`
check	`/diff` what changed · `/rewind` files, conversation or both · `/usage` cost and limits (`/cost`) · `/btw` a side question · `/export` · `/doctor` · `/help`
keys	Shift+Tab cycle the modes · Esc interrupt · Esc Esc rewind · Ctrl+O the transcript · Ctrl+B background a task · Ctrl+R history · Ctrl+C / Ctrl+D
prefixes	`/` a command or skill · `!` a shell command, its output into the session · `@` a file
modes	default (manual): reads only · acceptEdits · plan · auto: a second model, the classifier, reviews · dontAsk · bypassPermissions: isolated containers only

© Zhigang Feng Reproduction prohibited without permission

Four dials: what it reads, how hard it thinks, how much it may do unasked, what it costs.

Source: Claude Code 2.1.303: command help and documentation, read 18 October 2026.

Cheat Sheet: Files, Flags and Recovery

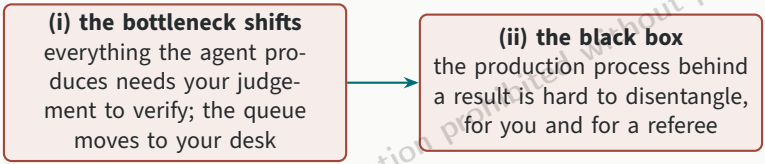
files it reads	`CLAUDE.md` the project brief (`AGENTS.md` is read in its place) · `~/.claude/CLAUDE.md` yours, every project · `CLAUDE.local.md` private, gitignored · `.claude/rules/*.md` · `.claude/settings.json`, `settings.local.json` permissions and hooks · `.claude/skills/<name>/SKILL.md` · `.claude/agents/<name>.md` · auto memory under `~/.claude/projects/`
flags on a cluster	`claude -c` continue · `-r <id>` resume · `--model` · `--effort` · `--permission-mode` · `--add-dir` · `--allowedTools` · `-p "<prompt>"` with `--max-budget-usd` · `--bg`, then `claude agents | attach | logs | stop` · `-w` a worktree · `--cloud`
recovery	connection cut → `tmux`, then `claude -r` in the same folder · window full → `/compact`, or `/clear` and a plan file · wrong turn → `/rewind` · broken setup → `claude doctor` · stale → `claude update`
habit	commit before you hand over the keyboard; read `git diff` before you believe a report

© Zhiqiang Feng
Redistribution prohibited without permission

State belongs in files; the session is a list you can resume, rewind or clear.

Source: Claude Code 2.1.303: command-line help and documentation, read 18 October 2026.

Two Dangers, Named Now



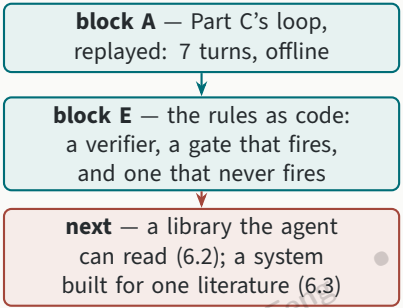
the evidence for (i) is Deck 1.1's survey of 637 scientists: the time the agent saved went to checking

- Both are real on the account in Part A; neither is an argument for going back.
- **The job is to make the advance work for us, and to name what it costs.**

Source: Codreanu, Imas, Mateos-Garcia et al. (2026), *AI in Science: Early Insights*, as read in Deck 1.1.

© Zhengyang Feng
Redistribution prohibited without permission

Your Turn: Lab Blocks A and E — Then the Library, Then the Basis



- Self-paced, after class, offline: a CPU, no account, a clean kernel. About forty minutes for the two blocks.
- Block A's model is a script, not a network call, so your transcript and the recorded one agree line for line.
- **Next: a library the agent can read (6.2), and a system built for one literature (6.3).**

Source: The session's lab, blocks A and E, and their recorded-results files.

© Zhigang Feng Redistribution prohibited without permission