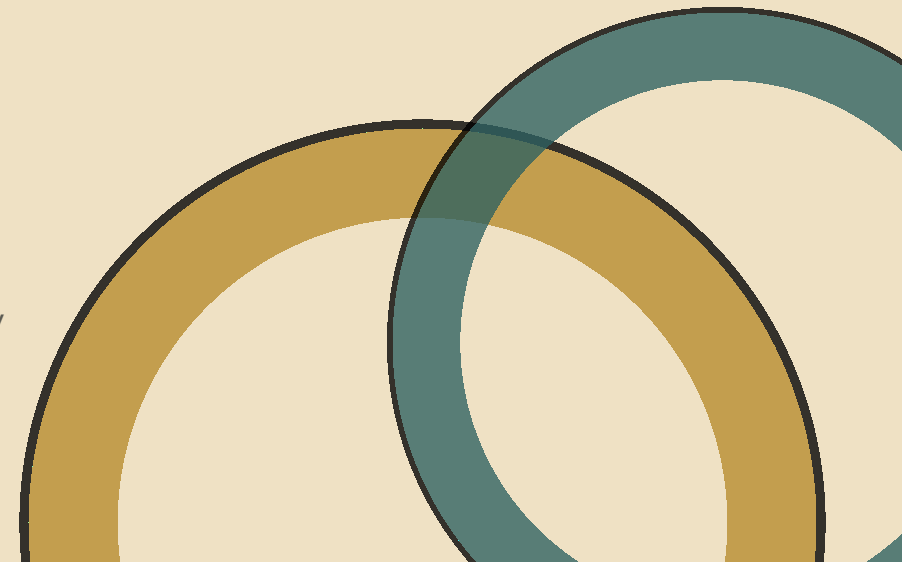


Planning, tools, and the action space

Plan globally, revise locally • a real verifier
beats self-judgment • tools shaped to the task



A goal nobody has turned into steps

“Migrate this service off the deprecated payments API.”

- Who invents the steps, and in what order?
- Who notices when a step went wrong?
- What can the agent actually *touch*?

Chapter 4 opens with this goal; the three questions are today's three parts

By the end of today you can

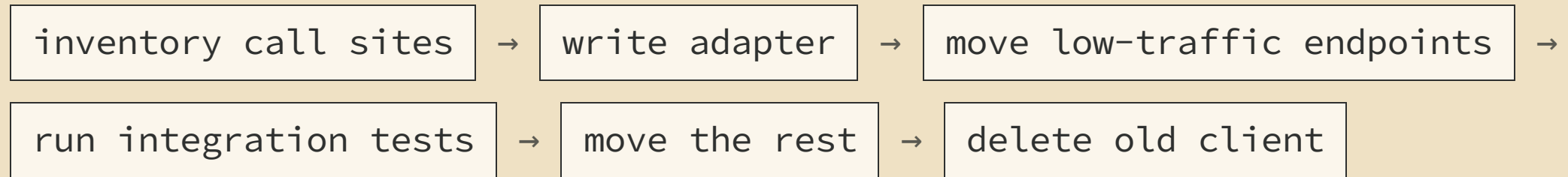
Four objectives

1. Contrast **plan-then-execute** with **interleaved planning** and say when each wins
2. Explain why “a real verifier beats the model grading itself”
3. Redesign a one-for-one API wrapping into a **task-shaped tool**
4. Write a tool description, schema and error message that pass the tool-contract linter

Part 1 • Chapter 4 (full book)

Planning, reasoning, self-correction

The list you would write on Monday

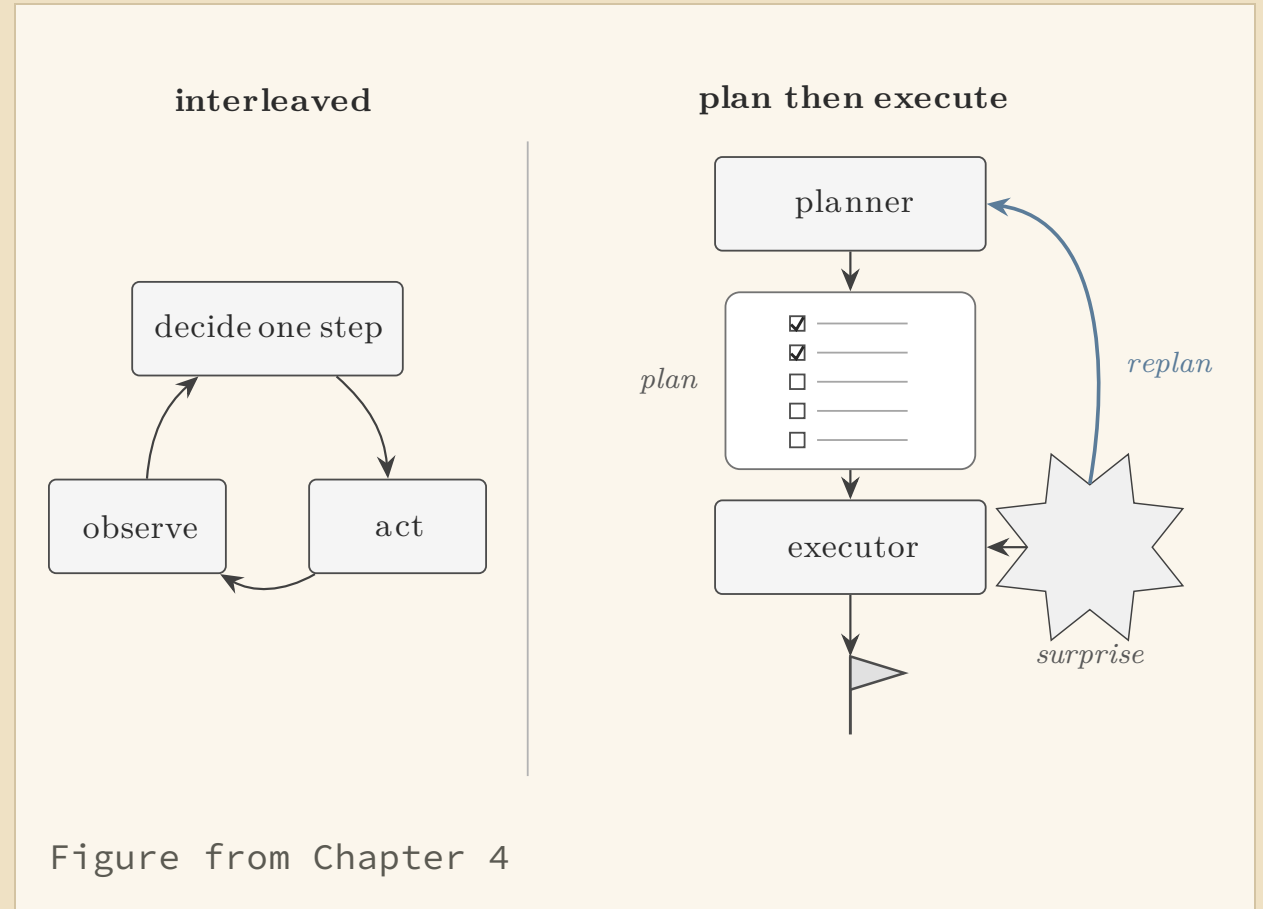


- Ask for it, shape the task so it falls out, or **hand the agent the plan**
- Lightest planning: the opening lines of a **chain-of-thought**
- A step out of order is destructive: its wreckage lands in the history

Chapter 4, “Task Decomposition and Planning” • steps as in the book

Interleaved vs plan-then-execute

- **Interleaved** (ReAct): one step at a time, chosen after each observation. Adaptive, *myopic*
- **Plan-then-execute**: a **planner** drafts the whole plan, an **executor** works through it
- Planning first attacks the **missing step**
- “Global coherence is a thing you buy.”



The part that separates finishing from flailing

“A plan with no replanning step is a guess with a schedule.”

Chapter 4, “Task Decomposition and Planning”

Replan on surprise

Return plan, completed steps and the surprise to the planner: finish, revise, or add steps

Write the plan down

A todo list in the history keeps the goal in view, and lets you approve it before execution

What does a wrong step cost to undo?

Task	Strategy
Short, cheap steps	No explicit plan; chain-of-thought is enough
Next move depends on the last result	Interleaved loop
Long, dependent, expensive mistakes	Plan-then-execute with replanning
Independent subtasks, latency matters	A dependency graph, run in parallel
You already know the steps	Draw the flowchart: a workflow

“Thinking, for a language model, is writing”

- Roughly the same computation per token, whatever is at stake
- **Chain-of-thought**: intermediate steps on the desk, before the answer
- First reach on anything multi-step; it costs one sentence

Order matters

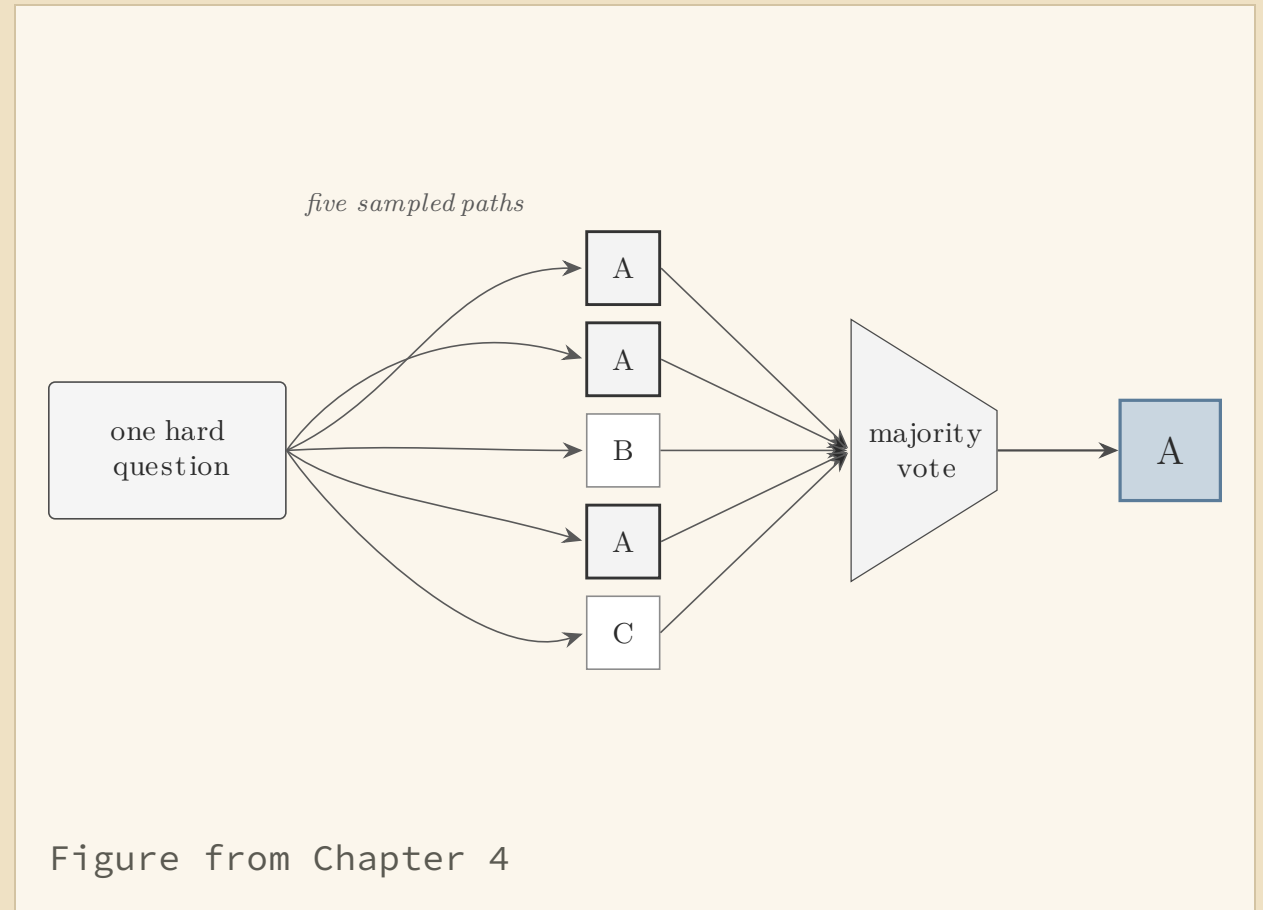
Text before the answer can **steer** it; text after it can only **rationalize** it

Read the trace as

“a map of places to dig, never as a proof of correctness”

Test-time compute and self-consistency

- **Test-time compute:** deliberate longer before answering; a dial to set per task
- **Self-consistency:** sample several paths, take the majority answer
- Evidence without an **oracle**; needs answers comparable for equality
- 10 samples = 10× the tokens, whether or not you show them



Predict before the next slide

You append: “Review your answer carefully. Identify any mistakes, and produce a corrected version.”

Averaged over many tasks, does the corrected version beat the original?

The experiment Chapter 4, “Reflection and Self-Correction”, hangs on

The answer has two columns

Works • flaws in the rendering

Style, tone, missing requirement

Draft–critique–revise: about 20 points better on average across seven tasks (Self-Refine)

Fails • flaws in the reasoning

Intrinsic self-correction

Models “struggle to self-correct their responses without external feedback, and at times, their performance even degrades”

- Reflexion’s famous gains: the post-mortem came *after* “task feedback signals”. The verdict walked in from outside
- Early “successes” stopped revising at the known answer: a smuggled-in **oracle**

Chapter 4 • Madaan, Huang, Shinn et al. (2023) • figures dated, mechanism durable

Why a good reasoner is a poor reviewer

Finding is hard; fixing is easy

Find the error

Weak

Same weights, same blind spots: “The model grades its own homework and likes what it sees.”

Fix a located error

Strong

“step four is wrong” → a well-posed repair, improved on every task tested

The division of labor: let the world find; let the model fix.

Chapter 4 • Tyen et al., “LLMs Cannot Find Reasoning Errors, but Can Correct Them Given the Error Location” (2024)

The sign the rest of the book points back to

“A real verifier beats the model grading itself.”

Chapter 4, “Reflection and Self-Correction”

Verifier: an external check that *exercises* the work: tests, compiler, schema validator, calculator, the query runs

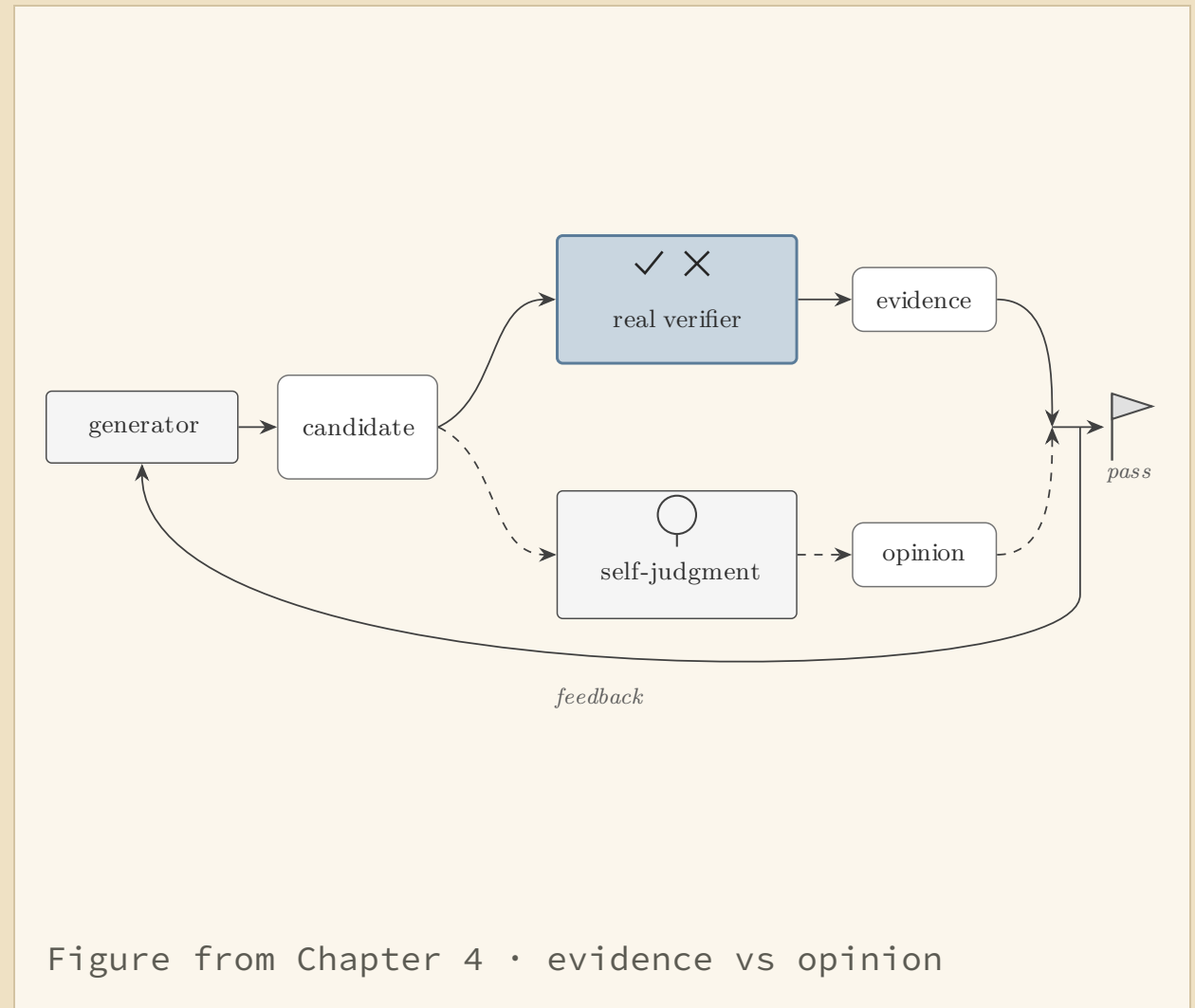


Figure from Chapter 4 • evidence vs opinion

When there is no compiler

A ladder of checkers

Rung	Its verdict is	Use it
Real verifier	a measurement of fact	wherever one exists
Independent critic (LLM-as-a-judge)	an estimate of quality	separate call, rubric, ideally another model
Self-review	opinion on the same desk	never voluntarily

Keep the checker separate from the maker. Feed failures back whole: message, stack trace, expected vs actual.

Part 2 • Chapter 5 (full book)

Tools and the action space

Take this literally

“The set of tools you expose is the complete inventory of what your agent can ever do.”

Chapter 5, opening

Traditional API

Deterministic code calling deterministic code; the caller cannot improvise

Tool

Deterministic code called by a **non-deterministic caller**: right tool with wrong arguments, wrong tool, invented tool

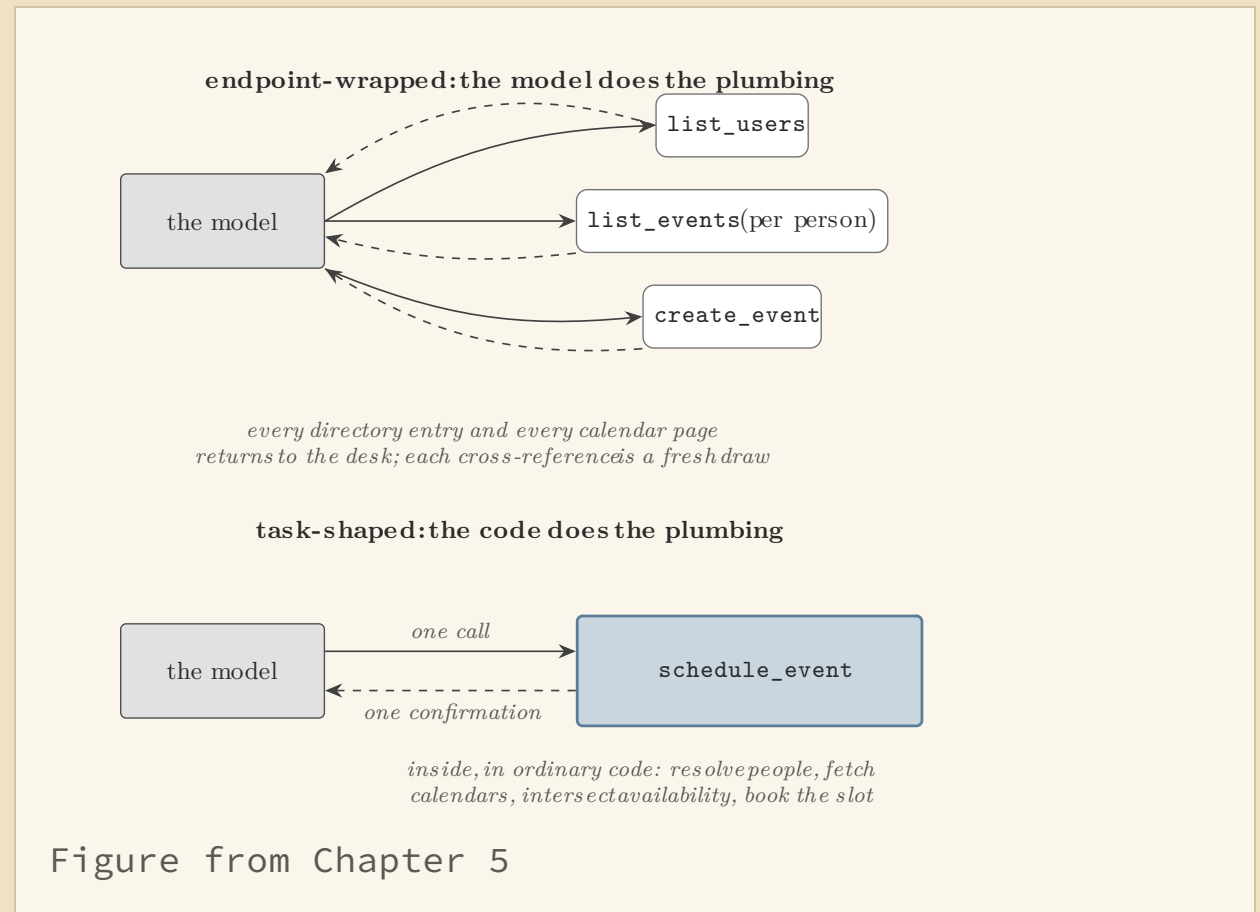
Worked example • “find thirty minutes for me and the design team”

Wrap the endpoints, or shape the task?

`list_users` + `list_events` +
`create_event`: directory and six
calendars onto the desk; the model
intersects by *reading*

`schedule_event(participants,`
`duration, window)`: code intersects;
the model sees one line back

The right unit for a tool is a task; the
endpoints are plumbing



The second reason one-for-one wrapping fails

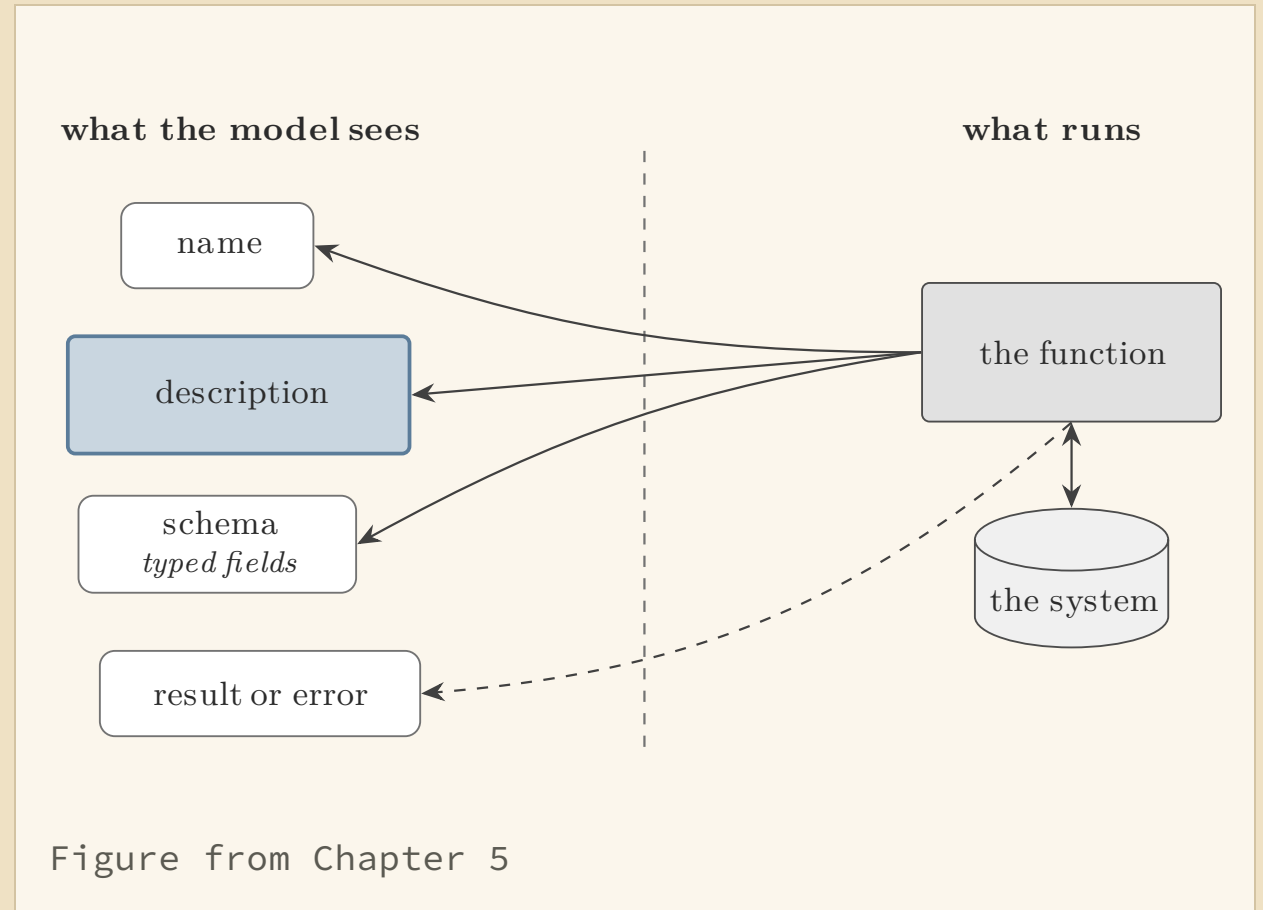
Too many tools confuse the model

- Every definition rides on the desk on **every pass**, and is an option to weigh
- Overlap (`read` vs `fetch`) presents a choice where none should exist
- Add capability without a tool: an **entry point**, or **delegation** to a subagent
- Unavoidably large? Namespace it: `crm_contacts_search`, `billing_invoices_search`
- Read transcripts; revisit the set on every model change. Subtraction is a design move

Chapter 5 • “More tools don’t always lead to better outcomes” (vendor guidance quoted there)

"Every one of these channels is a prompt"

- **Description:** for a capable new hire; when to use it, when not
- **Names:** `user` invites four readings; `user_id` one
- **Schema:** enums, required, no extra properties
- **Results:** the relevant slice; names over UUIDs; say when truncated



The error channel is a design surface

The highest-signal message a tool sends

Teaches nothing

Error 422

Corrects itself next pass

severity must be one of: low, med, high
(got "urgent")

Lab 2's structured error (Ch. 18)

```
{  
  "code": "INVALID_ARGUMENT",  
  "message": "severity must be one  
    of: low, med, high (got 'urgent')",  
  "retryable": false,  
  "hint": "ask the user which  
    severity they mean"  
}
```

Chapter 5, “The Tool Interface” • field shape from Chapter 18 (code, message, retryable, hint)

Make explicit what a human used to supply implicitly

- **Never block** on a prompt; fail fast when there is no terminal
- Destructive: confirm by default, `--yes` to bypass
- **Structured output** on stdout; diagnostics on stderr
- Help says *when* to use a command, and what comes next

Chapter 5 • “Crashes are tolerable; hangs are problematic” (field notes quoted there)

Assume every call will be retried

- **Idempotent:** a retried create returns the existing record
- `--dry-run` on every mutation
- Predictable vocabulary: `posts list`, `posts create`
- Harden inputs as if they were hostile

In class • teams of three • 20 minutes

Lint seven flawed tools, fix them, re-lint

1. Open the set in the browser linter: [seven flawed tool definitions](#) (also in your handout)
2. Fix every “Fix first” finding; then the “Should fix” ones
3. Find two flaws the linter *missed*; name the chapter rule each breaks
4. Re-lint; copy the result as Markdown and submit with your fixed JSON

read_logs

search

find_records

create_ticket

deleteUser

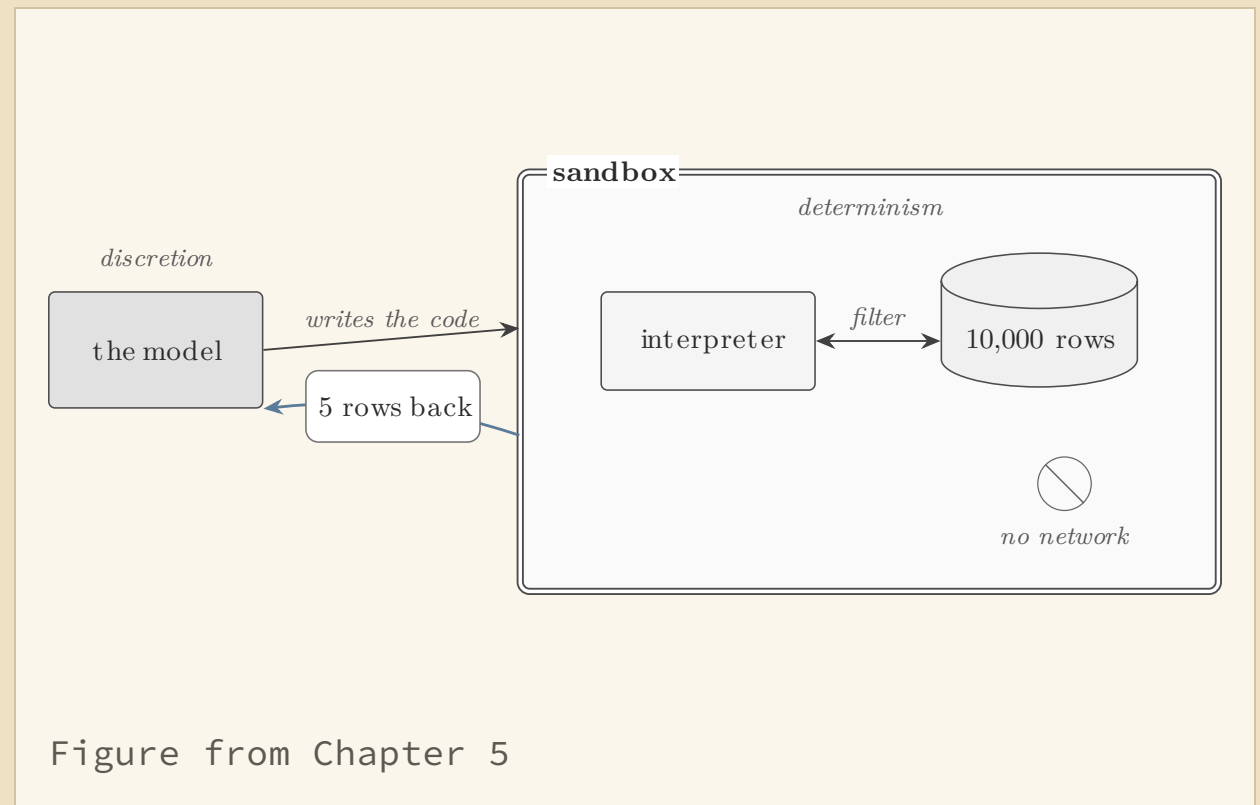
get_customer

run

aiagentsengineered.com/tools/tool-contract-linter/ • runs in the browser • no account, no API key

"Discretion on the outside, determinism on the inside"

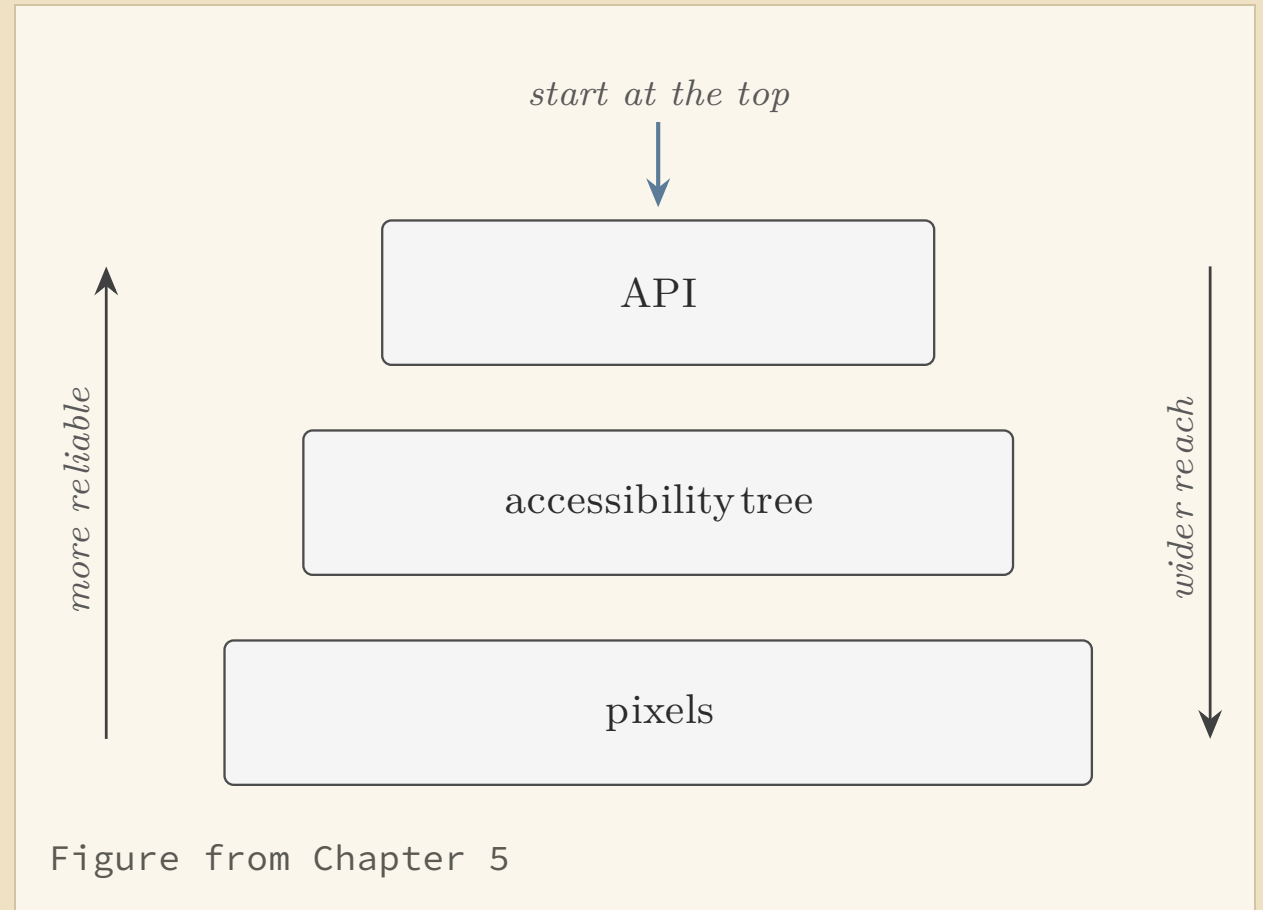
- One **meta-tool**, `run_code(source)`, composes all the rest
- 10,000 rows stay in the runtime; five lines come back
- **Sandbox:** own filesystem, capped CPU, memory and time; network denied by default; fresh per session
- A script that ran proves nothing about what it computed



When there is no tool end at all

Computer use: descend only as far as forced

- **API:** a deterministic contract
- **Structured browser:** the **accessibility tree**; click *by meaning*
- **Pixels:** universal, slowest, most fragile
- The page is input written by strangers: **indirect prompt injection**



Five things to keep

- **Plan globally, revise locally**; no replanning step means a guess with a schedule
- Thinking is writing: work before the answer, never after
- A real verifier beats the model grading itself: let the world find, let the model fix
- The action space is everything the agent can do; shape tools to tasks, keep them few
- Descriptions, results and errors are prompts; write them for a colleague who cannot see you

Before next week

Lab 2 and reading

Lab 2 • extend Lab 1

- `run_tests`: a real verifier
- `search_logs`: a task-shaped tool
- Every error: `code`, `message`, `retryable`, `hint`
- `dry_run` on the mutating tool
- Scripted-client test: recovery from a malformed argument

Reading for week 4

- Ch. 6, Skills, Protocols, and Interoperability `full book`
- Ch. 7, Managing the Context Window `full book`

This week's reading: Ch. 4 • Ch. 5 (full book) • no paid API needed for the lab

Engineering AI Agents • Week 3

Let the world find; let the model fix.

Slides from the teaching kit of *AI Agents, Engineered* by Enrique Gutiérrez, CC BY 4.0 • aiagentsengineered.com/teach/

Reuse, adapt and translate with attribution • creativecommons.org/licenses/by/4.0/

