上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Lecture 35: Tool-Using LLM Agents

Tao Huang

John Hopcroft Center, School of Computer Science, Shanghai Jiao Tong University

https://taohuang.info/cs3317

https://oc.sjtu.edu.cn/courses/89538

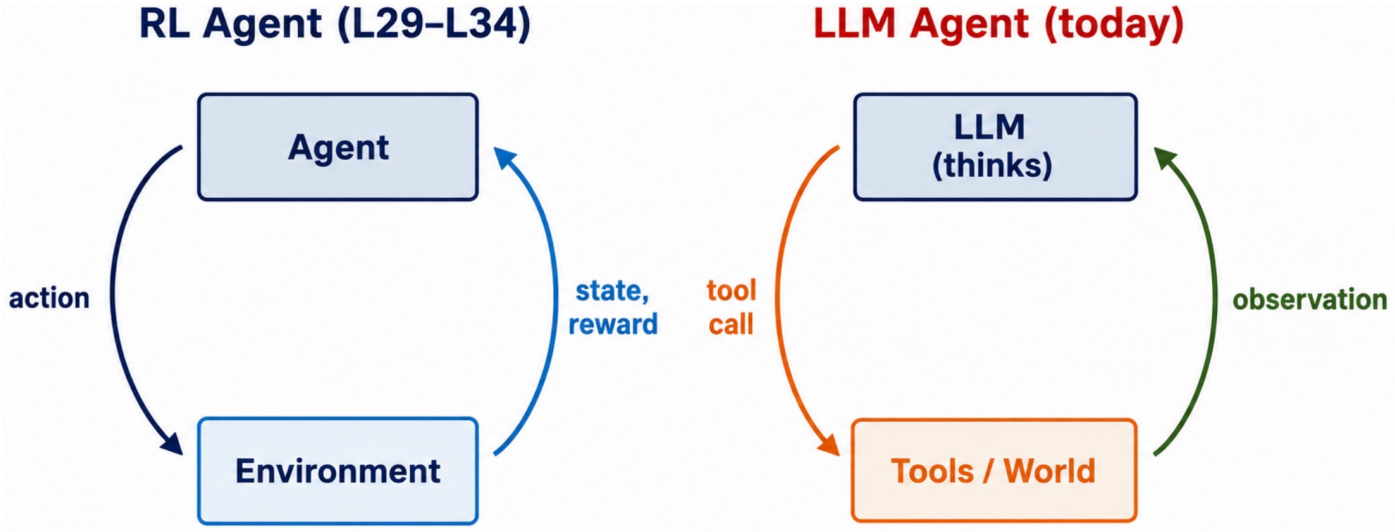
AI tools assisted in generating some figures in these slides. All such content has been reviewed, and the instructor is responsible for its accuracy.

From Predicting Text to Taking Action

L34: we learned a model and planned inside it.
L35: the model is GPT — and "planning" is called "reasoning".

- RL gave us agents that **act in an environment to maximize reward** (L29–L34).
- An LLM just **predicts text**. Today: what happens when the text-predictor takes actions.

By the end, you'll know exactly what Cursor / Claude Code is doing every time you hit "accept."



The Gap Between Knowing and Doing

- A plain LLM can write you a travel itinerary in seconds.
- But ask it “what's the weather in Tokyo right now?” — it cannot know.
- Its knowledge is frozen at training time, and it cannot do anything.

How do we turn a model that only predicts text into one that can search, compute, book, and act?

That gap — between knowing and doing — is exactly what the agent abstraction closes.

Objectives

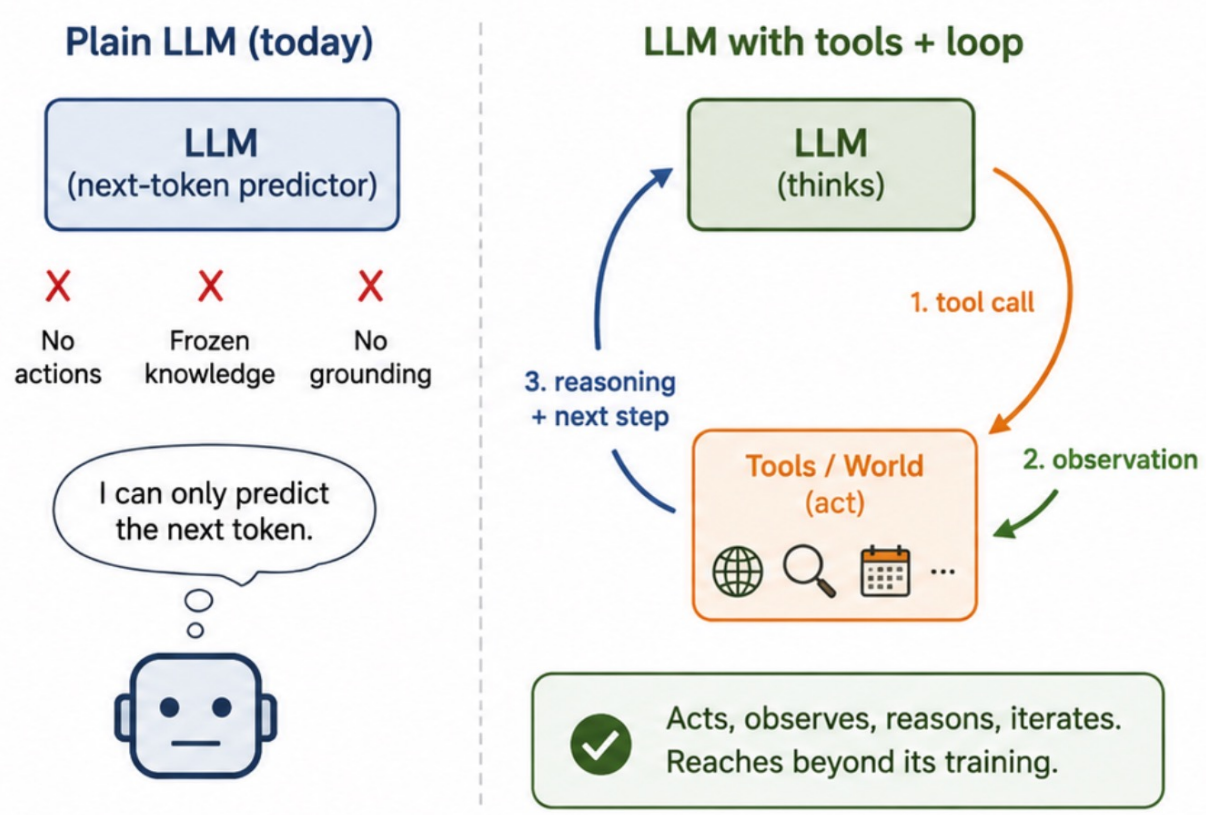
By the end of this lecture, you should be able to

- Explain why a frozen, passive LLM needs tools.
- Trace one ReAct loop (thought → action → observation) on a concrete query.
- Map the agent you use daily (Cursor / Claude Code) onto that loop.
- Distinguish RAG from tool use, and both from fine-tuning.
- Evaluate where agents break, and whether a given task suits one.

1. From Text Predictor to Agent

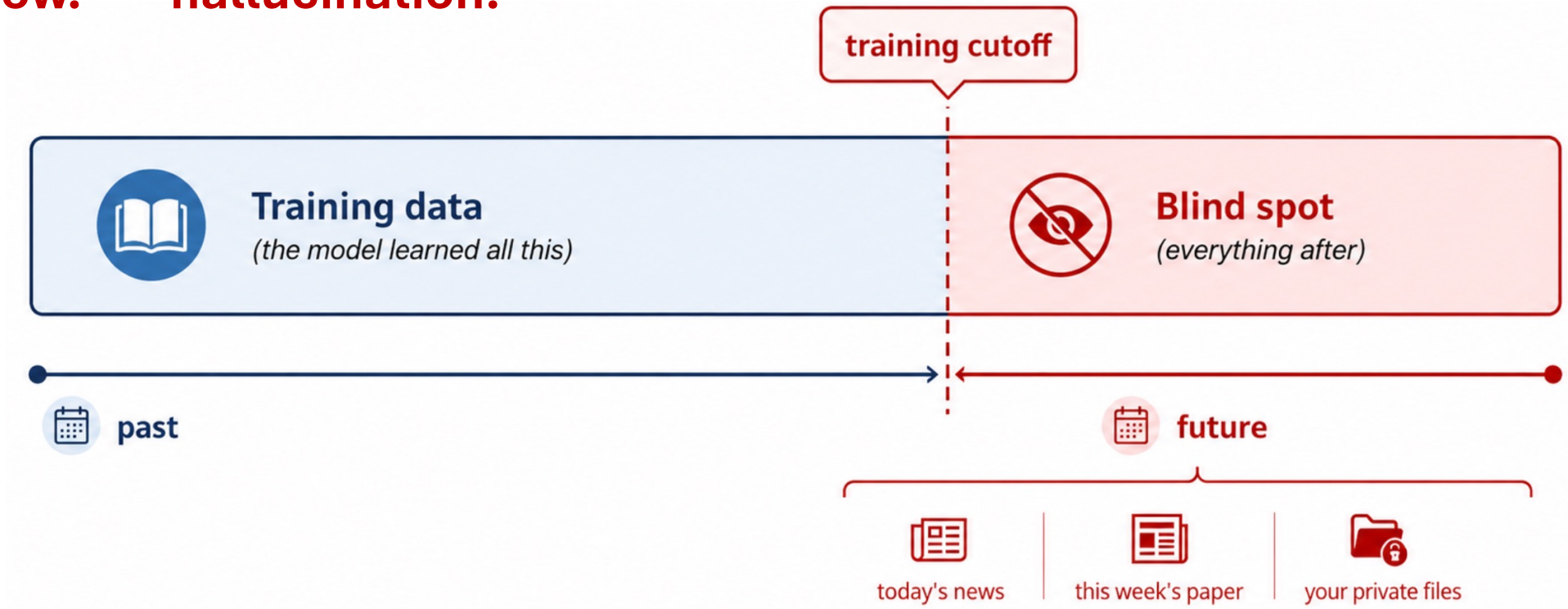
What a Plain LLM Cannot Do

- **Situation:** We have an extraordinarily capable next-token predictor (L27).
- **Complication:** Three hard limits — frozen knowledge, no actions, no grounding.
- **Question:** *Can we fix these without retraining the model every day?*
- **Answer:** Give it tools and a loop. Don't change the brain — change what it can reach.



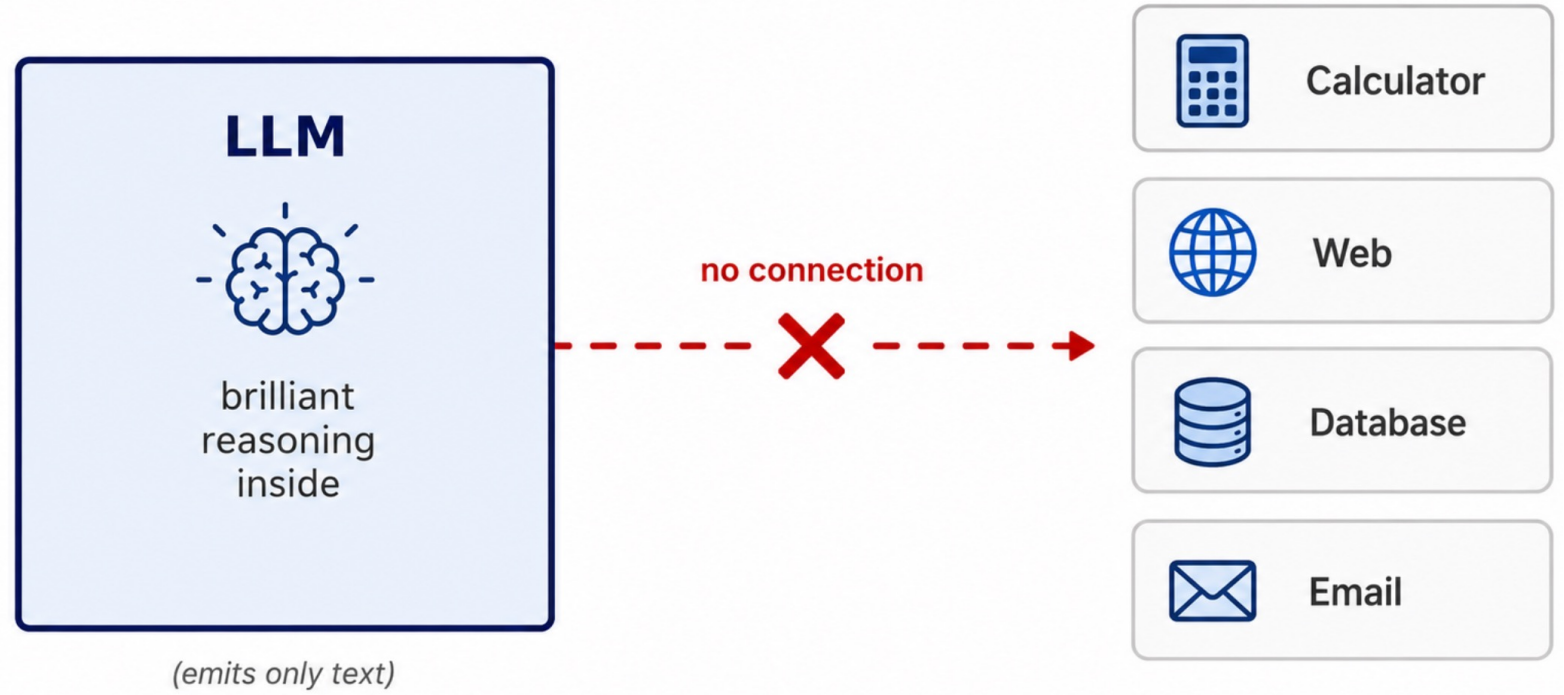
Limit 1 — Frozen Knowledge

- Model weights freeze at the training cutoff. Anything after is invisible.
- Today's stock price, papers from last week, your private files.
- **It will often confidently make something up rather than say "I don't know." — hallucination.**



Limit 2 — No Hands

- The model emits text. That's all.
- It cannot send an email, run a Python script, or query a database.
- **A brilliant strategist locked in a room with no phone.**
- Even perfect reasoning is useless if it can't reach the world.



Limit 3 — It Can't Check Itself

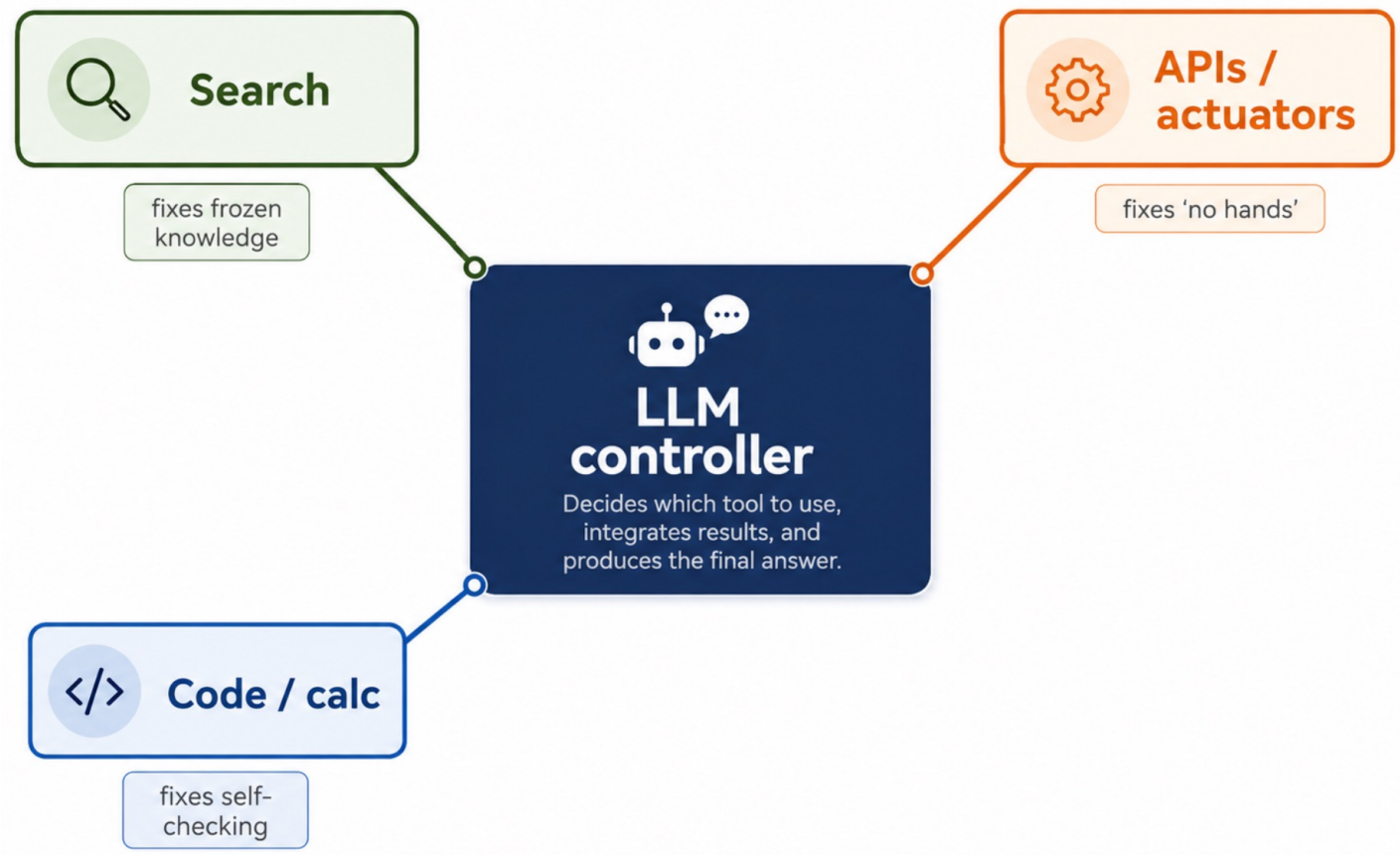
- LLM arithmetic on big numbers is unreliable — it pattern-matches, it doesn't compute.
- It cannot verify a claim against a source — there's no source to consult at inference.
- It can write code that looks right but won't know if it runs — until something executes it.

The model is fluent, not grounded. Fluency is not truth.

Ask it to multiply two 6-digit numbers → often wrong. Give it a calculator tool → always right.

The Fix: Augmentation, Not Bigger Models

All three limits share one fix: let the model call external tools and feed the results back in.



The LLM becomes a controller that decides which tool to use and when — not the thing that does everything.

Think: Tool or Train?

For each task, would you give the model a tool, or retrain it?

(a) Answer questions about your company's internal wiki

(b) Always respond in a polite, formal tone

(c) Tell me yesterday's football scores

Discuss with your neighbor before we reveal.

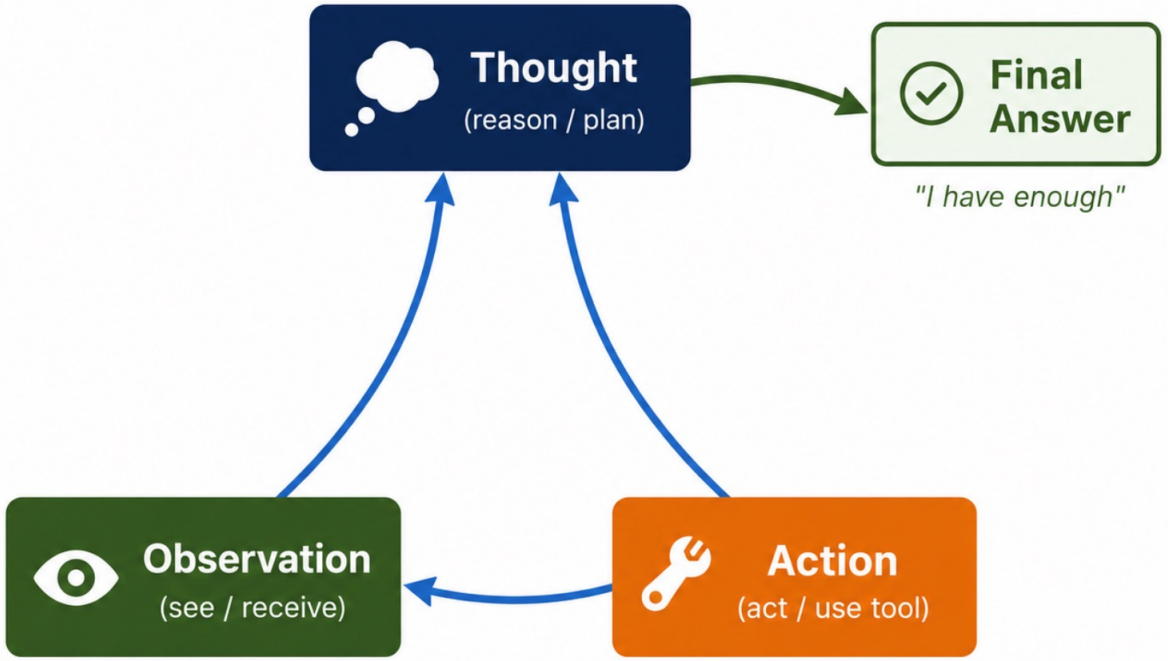
2. The Agent Loop: Reason + Act

One Tool Call Isn't Enough

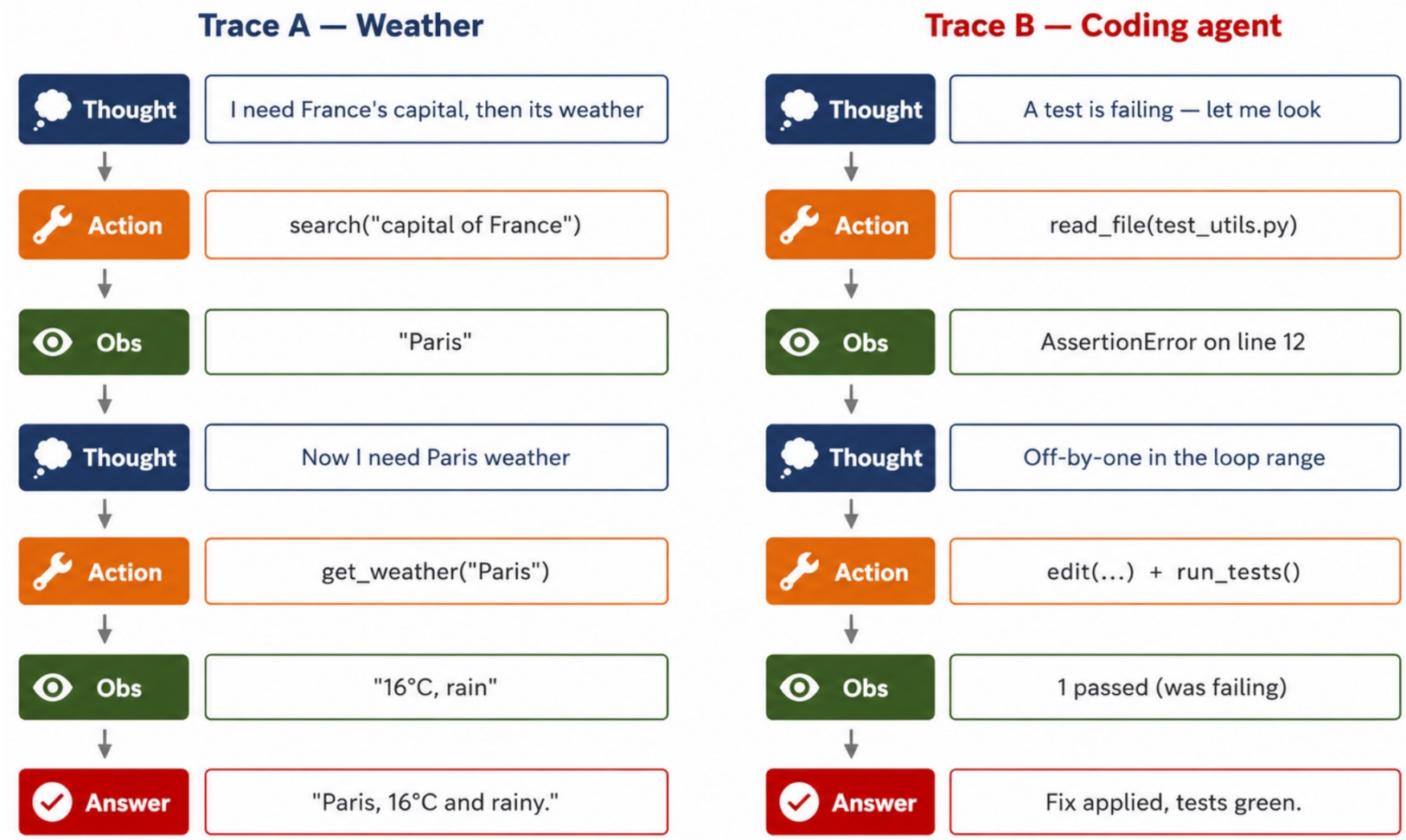
- **Situation:** We can let the model call a tool once.
- **Complication:** Real tasks need many steps, each depending on the last result.
  "Book the cheapest flight to a city with good weather next week."
- **Question:** *How does the model pick the next action from what it just observed?*
- **Answer:** Interleave reasoning and acting in a loop — the ReAct pattern.

ReAct: Thought → Action → Observation

- **1. Thought** reason about what to do →
  2. Action emit a tool call →
  3. Observation read the result.
- Repeat until done. Yao et al., 2022. The reasoning is written out in text — **no new training.**



Worked Example



Same loop. The weather agent and your coding agent are the same machine.

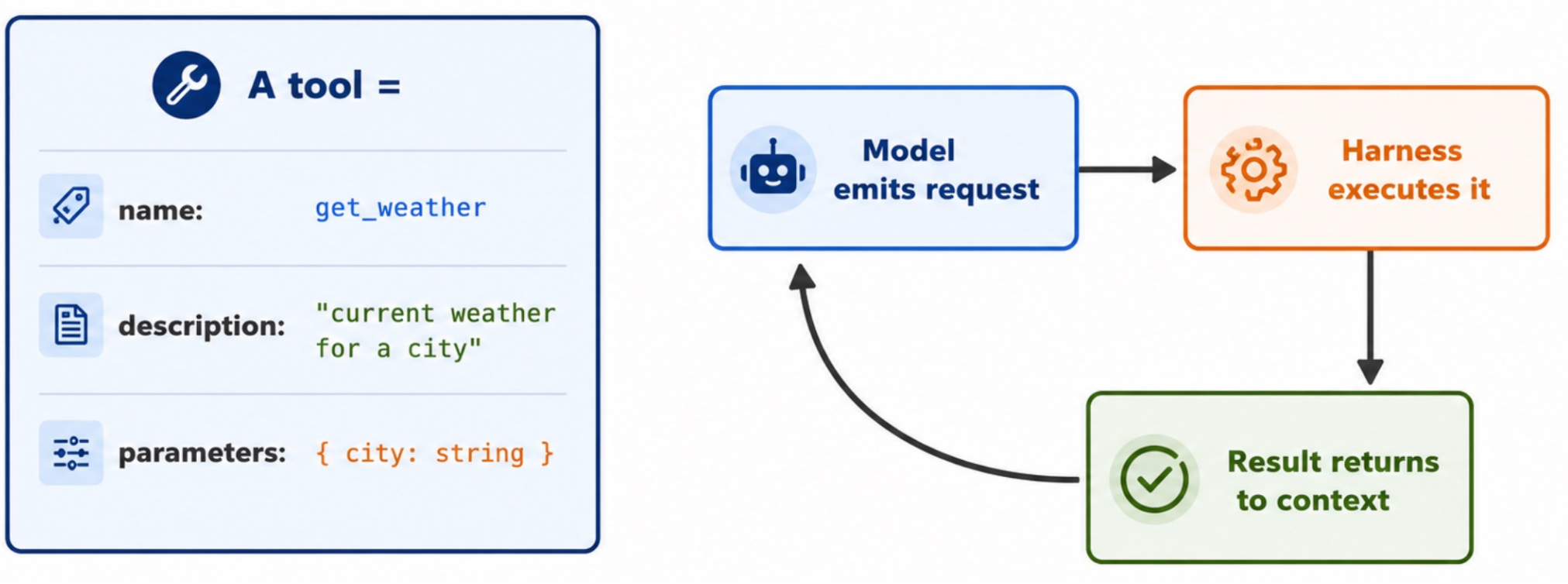
Why Writing the Thought Down Helps

- **Forcing a "Thought" before each action measurably improves reliability.**
- It's chain-of-thought applied to actions.
- **It also makes the agent debuggable — you can read exactly why it did what it did.**

But it's not magic: the thoughts can be confidently wrong too.
They expose the reasoning — they don't guarantee it.

Anatomy of a Tool

- Mechanically, a tool is just a name, a description, and an argument schema.



The model doesn't run the tool
— it emits a request; an external harness runs it and returns the result.

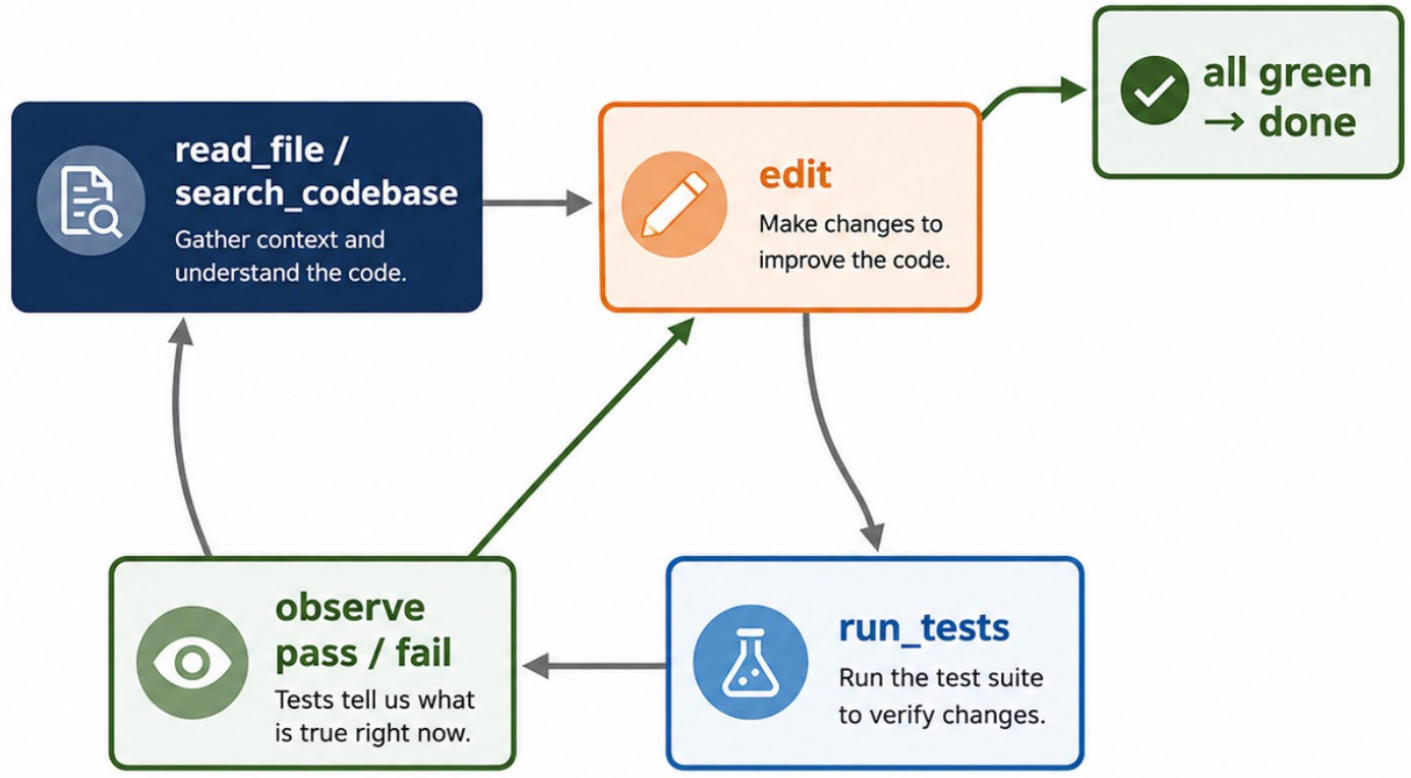
Function Calling: How Tool Use Got Reliable

- Early agents parsed tools out of free text — brittle and error-prone.
- **Modern models are trained to emit structured tool calls ("function calling").**
- The output is a structured request the system can guarantee is well-formed.
- **The 2023–24 reliability jump came from a better interface — not a smarter loop.**

Anatomy of a Coding Agent

The loop you watch every day:

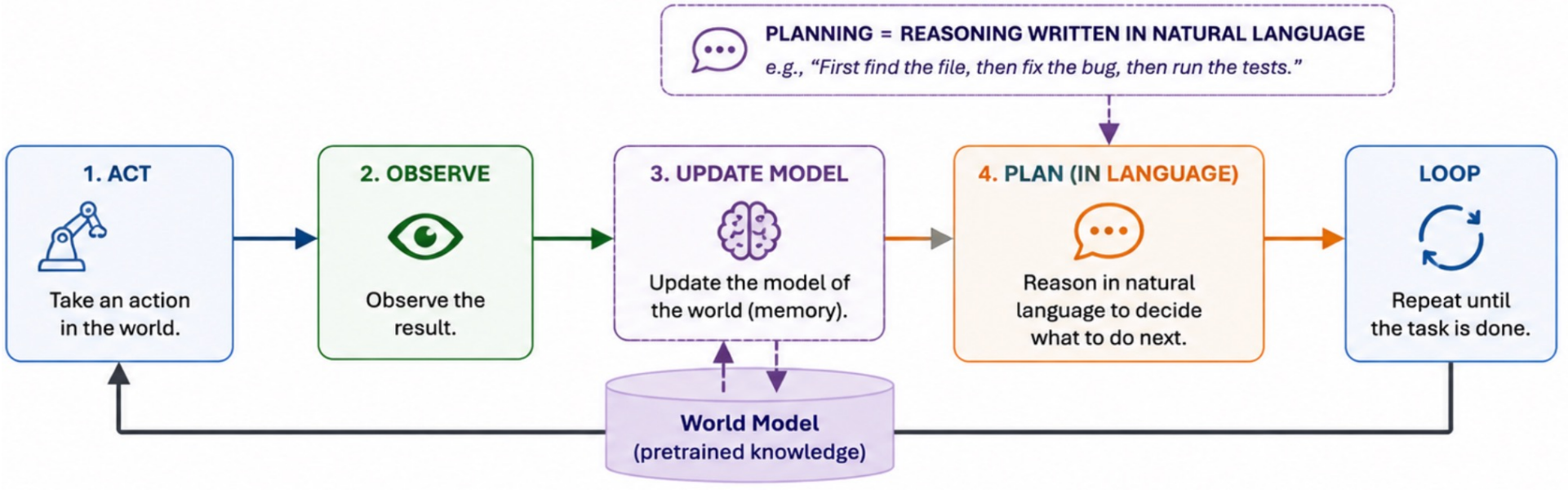
- **Tools:** read_file, search_codebase, edit, run_tests.
- **Loop:** read files → propose edit → run tests → observe pass/fail → fix → repeat.
- *It's a ReAct loop with real tool names — nothing new under the hood.*



Coding agents are the best-behaved agents today: tests are ground-truth observations.

Callback: This Is Dyna-Q, in Language

L34's Dyna loop: act → observe → update model → plan.
ReAct: think → act → observe → repeat.



Same skeleton. Two differences:

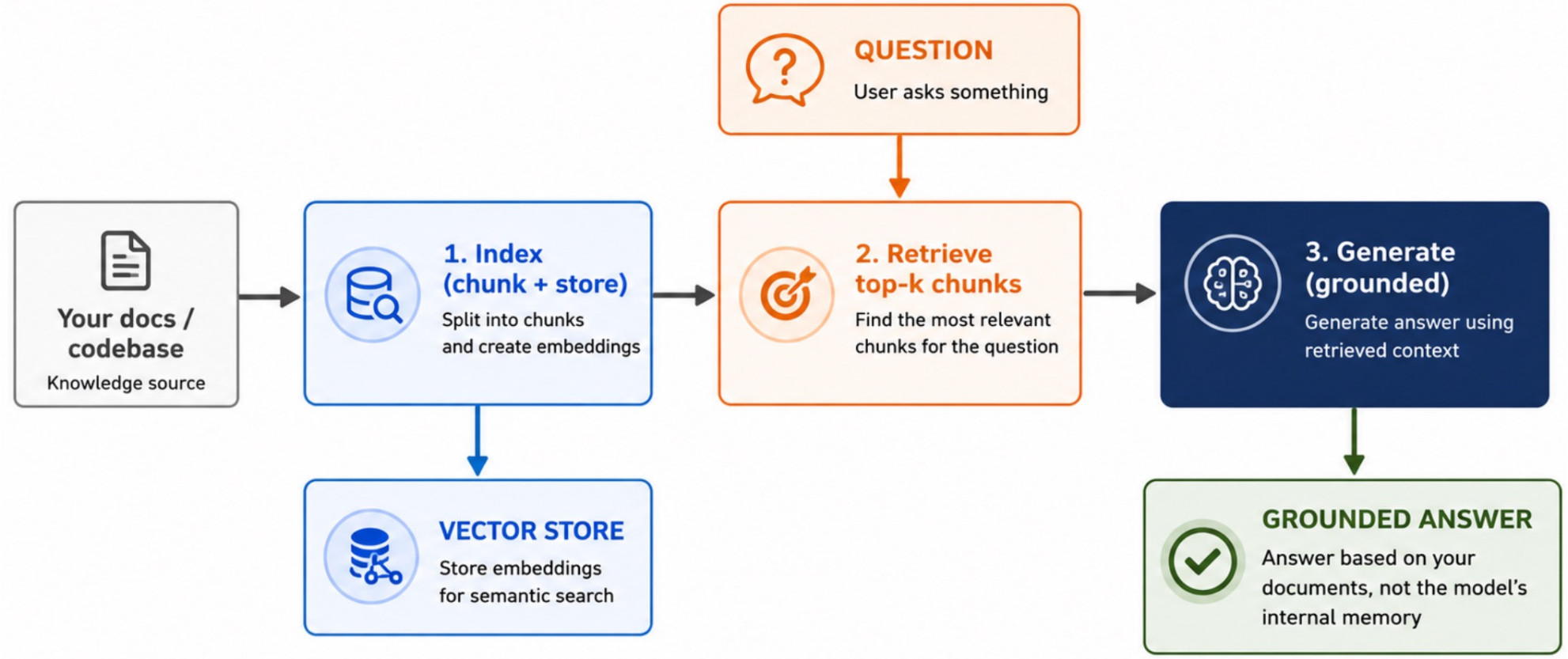
- The "model of the world" is now the LLM's pretrained knowledge.
- "Planning" is reasoning written in natural language.

3. Connecting Agents to Knowledge

The Knowledge Problem

- **Situation:** Search fixes “frozen knowledge” for public facts.
- **Complication:** What about your knowledge — company docs, a textbook, private files?
  You can't web-search those, and retraining on them is slow and expensive.
- **Question:** *How can an agent answer from a collection it never trained on?*
- **Answer:** Retrieval-Augmented Generation — retrieve, paste into the prompt, answer.

RAG in Three Steps



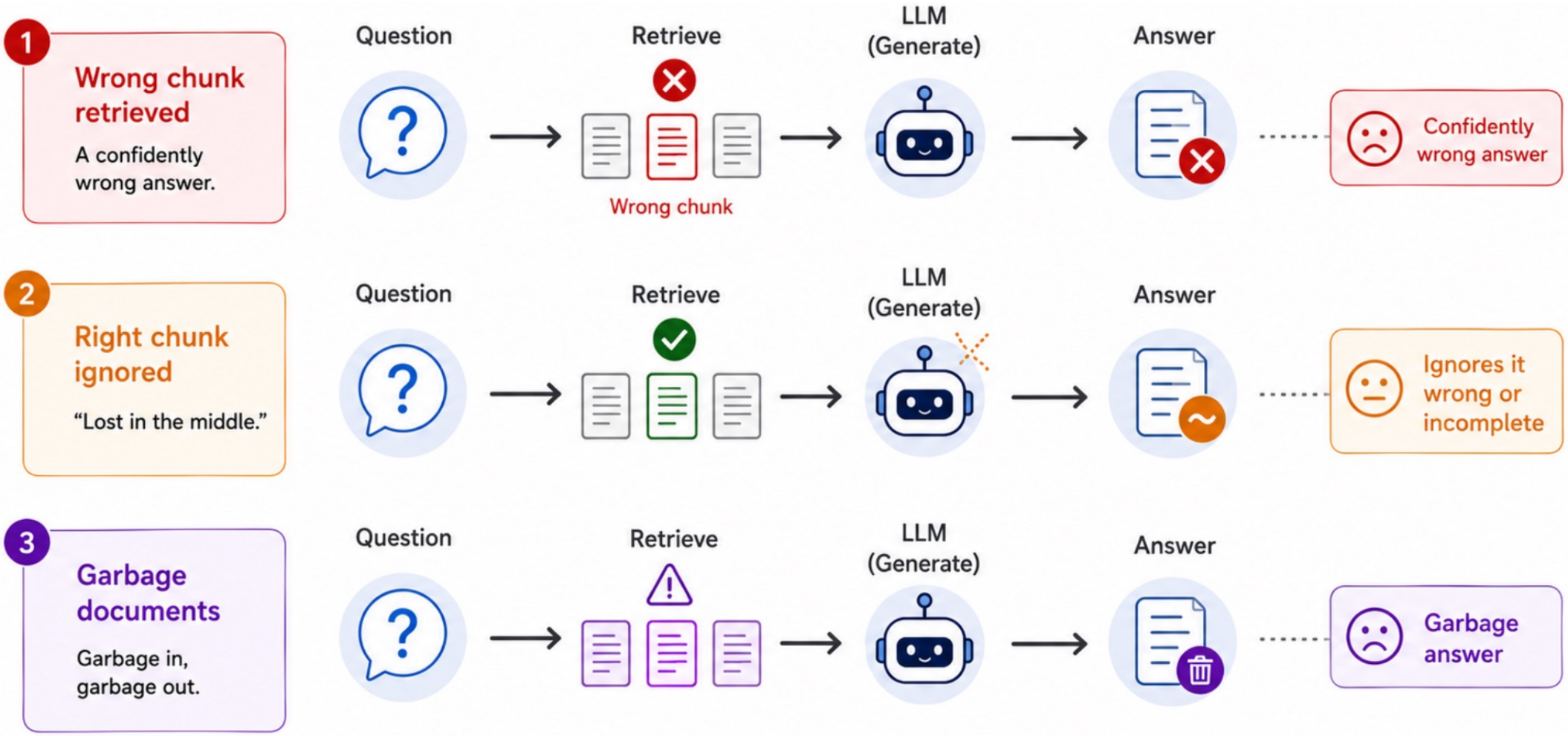
- **Index** (chunk + store by meaning) → **Retrieve** (top-k) → **Generate** (grounded answer).
- This is how Cursor "knows" about a function defined in a file it never had open — **RAG over your codebase.**

Why RAG Beats “Just Fine-Tune It”

	RAG	Fine-tuning
Update knowledge	edit the documents	retrain the model
Cost to add a fact	~free	$$$ expensive
Shows its sources	yes (the chunks)	no
Good for	facts, documents	behavior, style, format

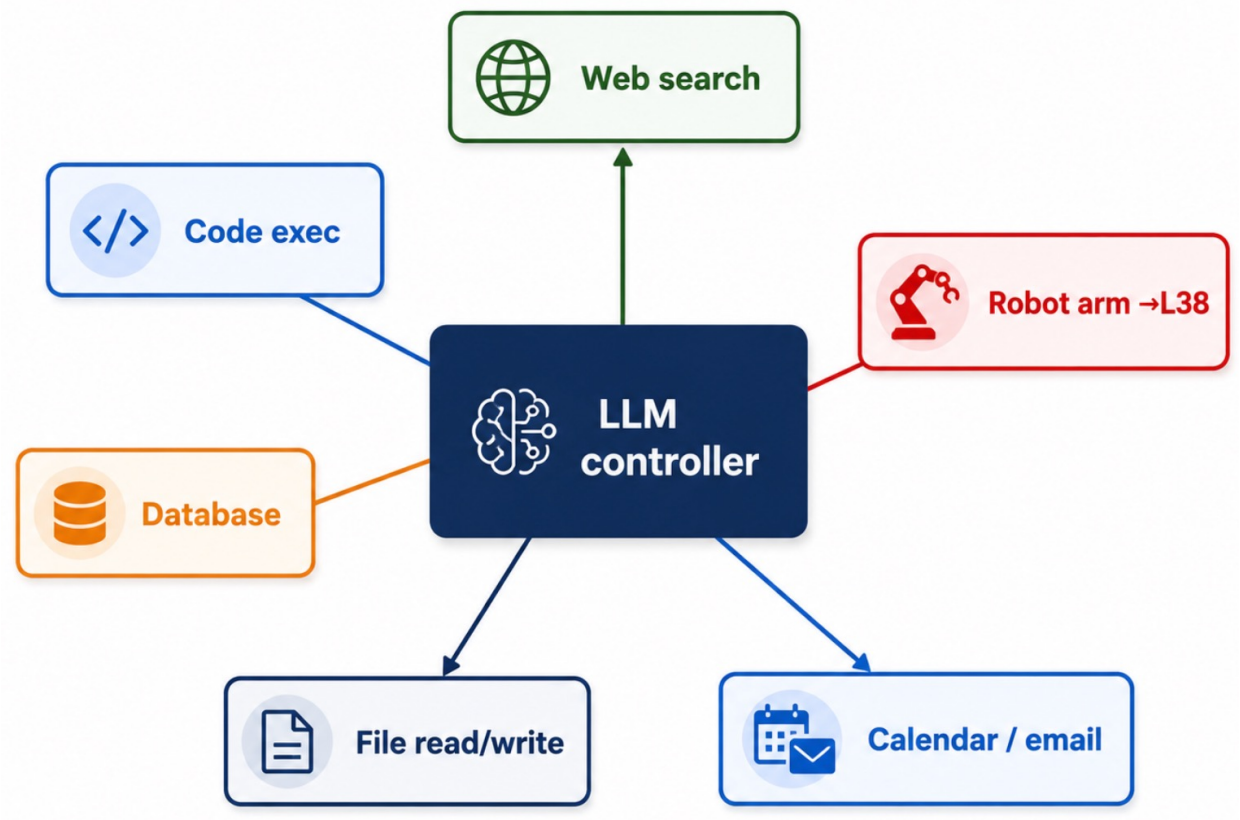
RAG is the default for knowledge; fine-tuning is for behavior.

RAG's Failure Modes



Grounding the model in a source is necessary — not sufficient.

The Modern Tool Ecosystem

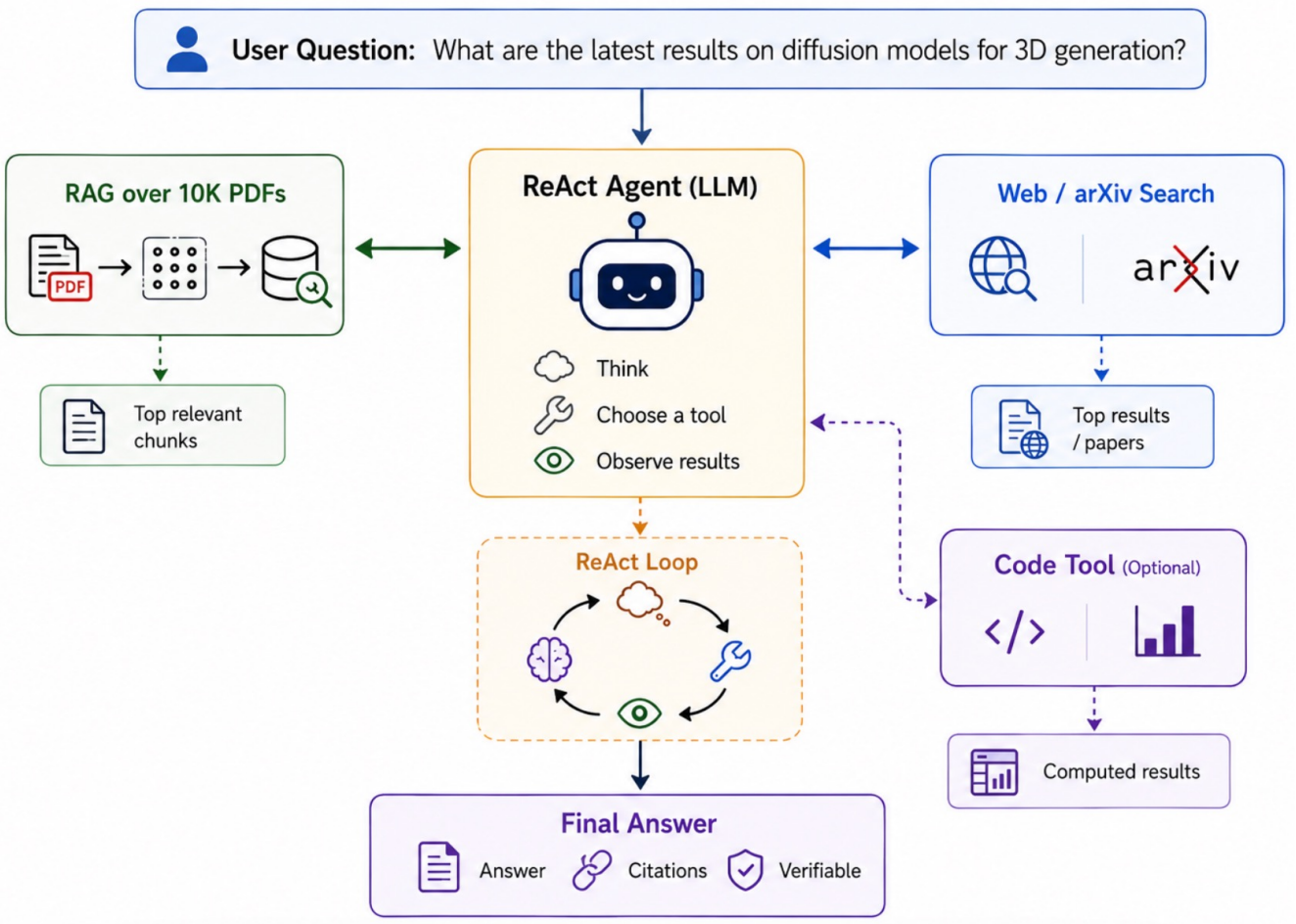


- A coding agent's toolset is this list pointed at a repo — read / write / run / grep.
- Emerging standard: **MCP** — a common "plug" so any model can connect to any tool.

Think: Design an Agent

You're building a research assistant that answers questions about 10,000 internal PDFs and can also check today's arXiv. Which components does it need?

Sketch it with your neighbor.



4. Do Agents Actually Work?

The Demo-to-Reality Gap

- **Agents are spectacular in demos and fragile in production. Why?**

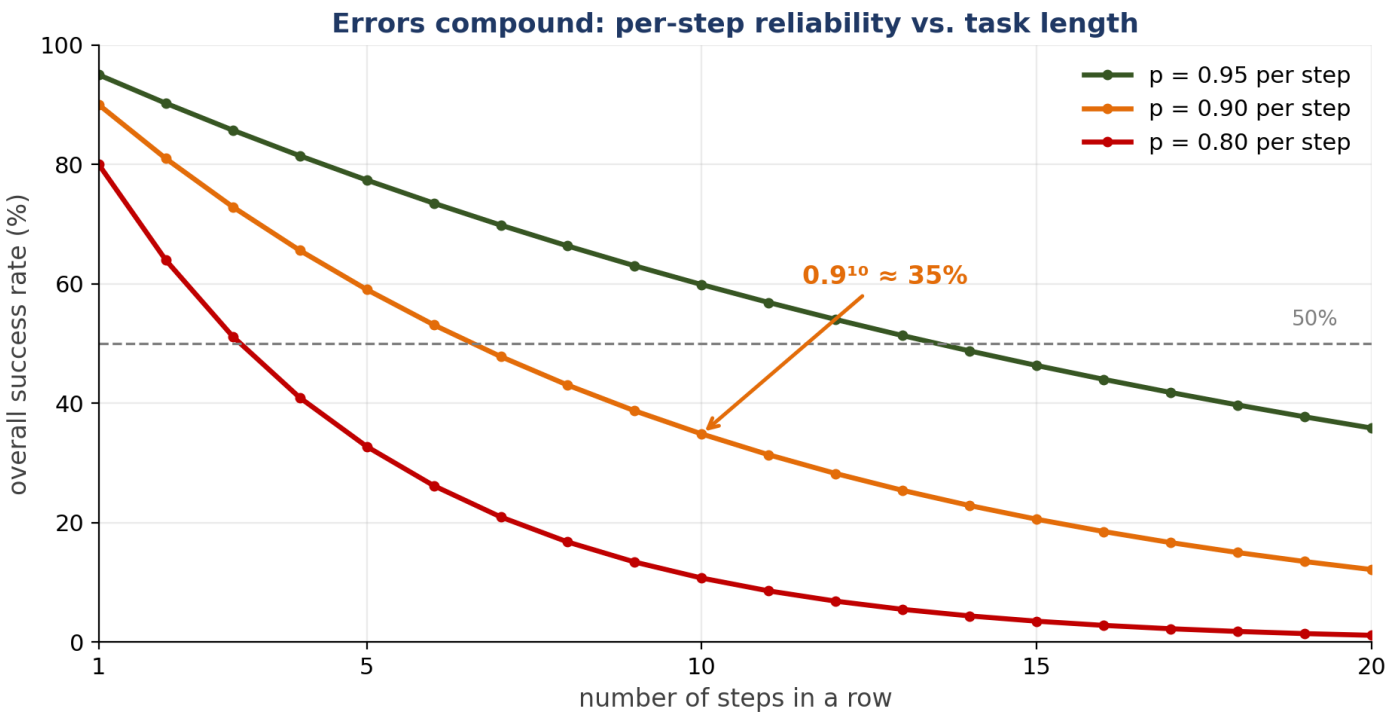
Each step works 90% of the time. Ten steps in a row: $0.9^{10} \approx 35\%$.

- Errors compound. One wrong observation early derails everything after.
- **Coding agents resist this: tests catch the error each step, so it can't compound silently.**
- Verifiable environments are the exception that proves the rule.

This is the defining open problem of agents in 2026 — and the setup for L36.

Four Ways Agents Fail

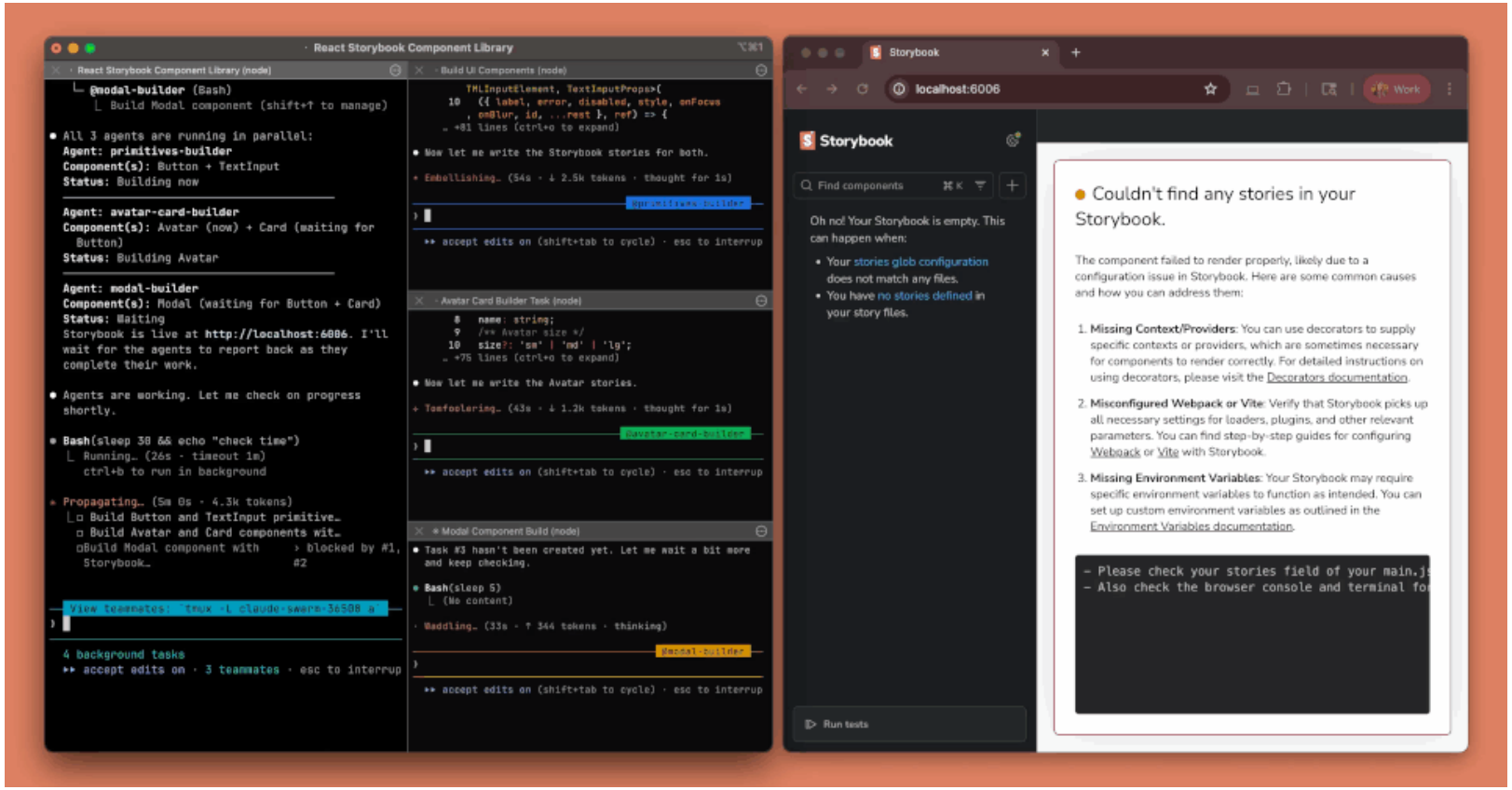
- **Error compounding**
  — the 0.9^n multiplication.
- **Hallucinated tool calls**
  — invents a tool or argument that doesn't exist.
- **Stuck in loops**
  — repeats the same failed action.
- **Cost & latency**
  — every step is a full model call; long loops are slow.



What Real Agent Products Do About It

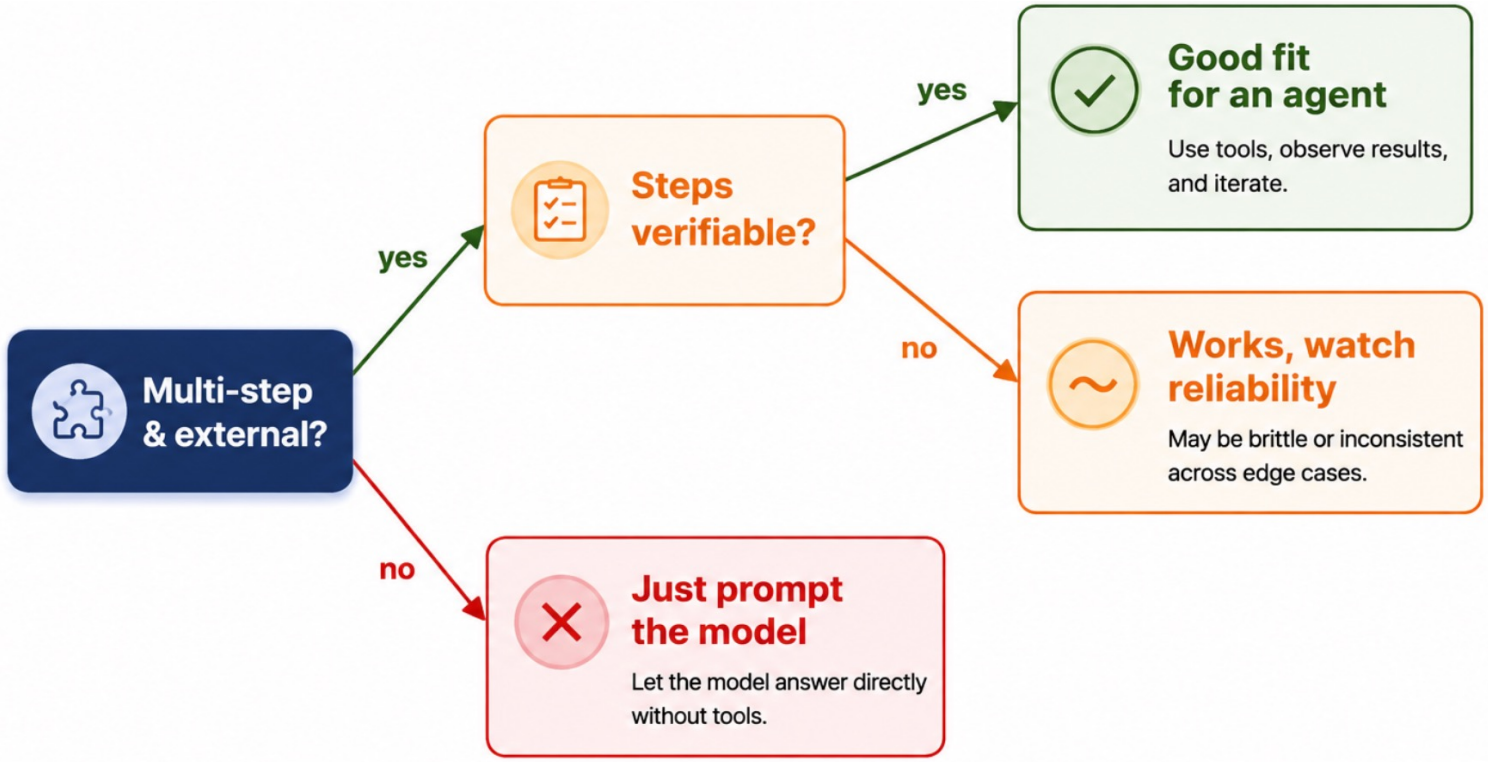
2024–26 examples you use:

- coding agents (Claude Code, Cursor, Copilot)
- deep-research agents
- computer-use agents



Wins are real but narrow: agents shine where steps are verifiable (code runs or it doesn't), and struggle where they aren't.

When Should You Use an Agent?



- **Good fit:** multi-step, needs live/external info, steps verifiable, tolerates latency.
- **Bad fit:** single-shot Q&A, millisecond latency, zero error tolerance.

Summary

- A plain LLM is frozen, handless, and ungrounded — tools fix all three.
- ReAct (think → act → observe → repeat) turns a predictor into an actor — no new training.
- RAG grounds the model in external knowledge; fine-tuning is for behavior.
- The agents that work best are the verifiable ones — which is why your coding agent is the most reliable agent you own.
- The core unsolved problem is error compounding over long horizons.

Frontier: Where This Is Going

- Computer-use agents that operate a screen like a person.
- Multi-agent teams that divide work and check each other.
- Agents that learn from their own past traces.

The field moves fast and over-promises faster.
Judge by long-horizon reliability, not by demos.

Next: When the Loop Falls Apart

L35 gave the agent a loop. But the loop falls apart over long horizons.

Two questions for next lecture:

- How does an agent stay coherent over dozens of steps without drifting?
- How does it remember what it did 50 steps ago?

L36: Long-Horizon Reasoning & Memory.