

AI ENGINEERING · PART II · 20-SLIDE SUMMARY

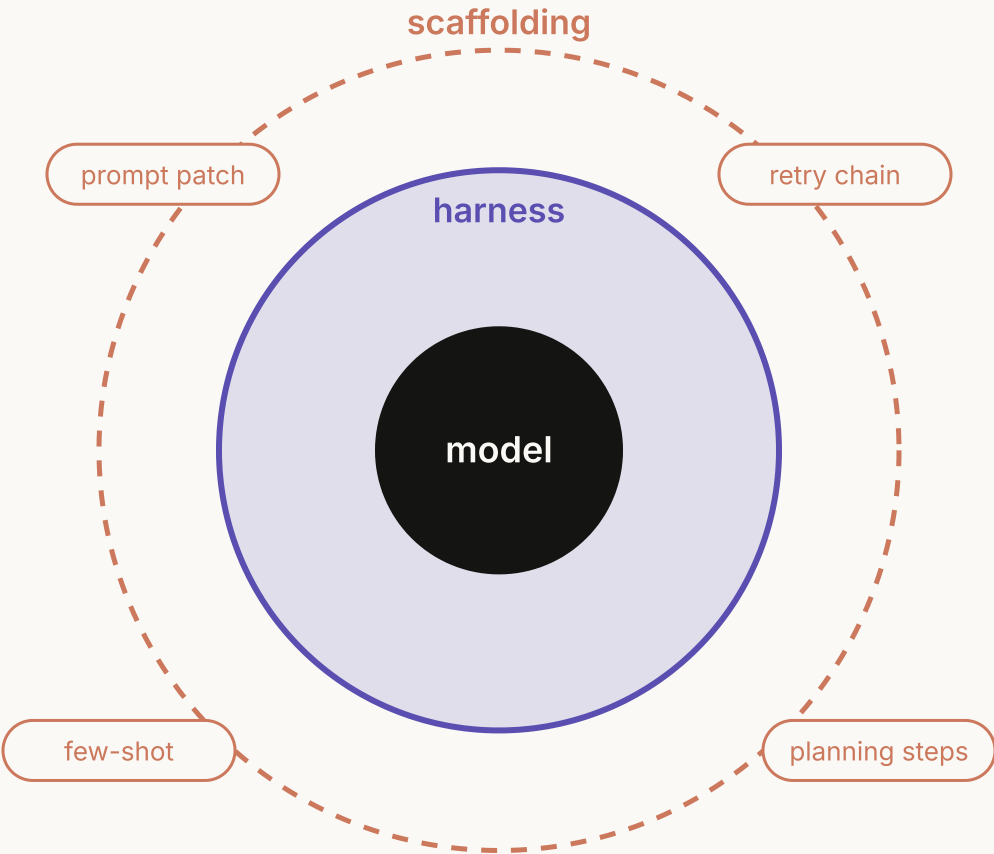
The art of AI engineering — Part II

The harness around the model, plus security, observability, delivery and continuous improvement: what makes it reliable in production.

Building, securing and operating the harness

PART 1 · SLIDES 3-6 Harness design 03 Harness vs scaffolding 04 Reference architecture 05 Build sequence 06 Make-or-buy options	PART 2 · SLIDES 7-12 Security 07 Threat landscape 08 Classical vs AI security 09 Attack surface by component 10 Attack taxonomy 11 Case study: OpenAI-Hugging Face incident (July 2026) 12 Defence controls	PART 3 · SLIDES 13-19 Delivery and operations 13 Release model 14 Delivery principles 15 Testing strategy 16 Release pipeline 17 Monitoring and drift 18 AI across the delivery lifecycle 19 Model upgrade process
--	--	---

One layer stays. The other should expire.



HARNESS · LOAD-BEARING

Exists because the model is a text function

tool interface

permissions, sandbox

control loop, when to stop

state and recovery

the grading gate

SCAFFOLDING · COMPENSATORY

Exists because *this* model can't hold the task

prompt patches for one failure mode

decomposition the model can't do

self-verification, retry chains

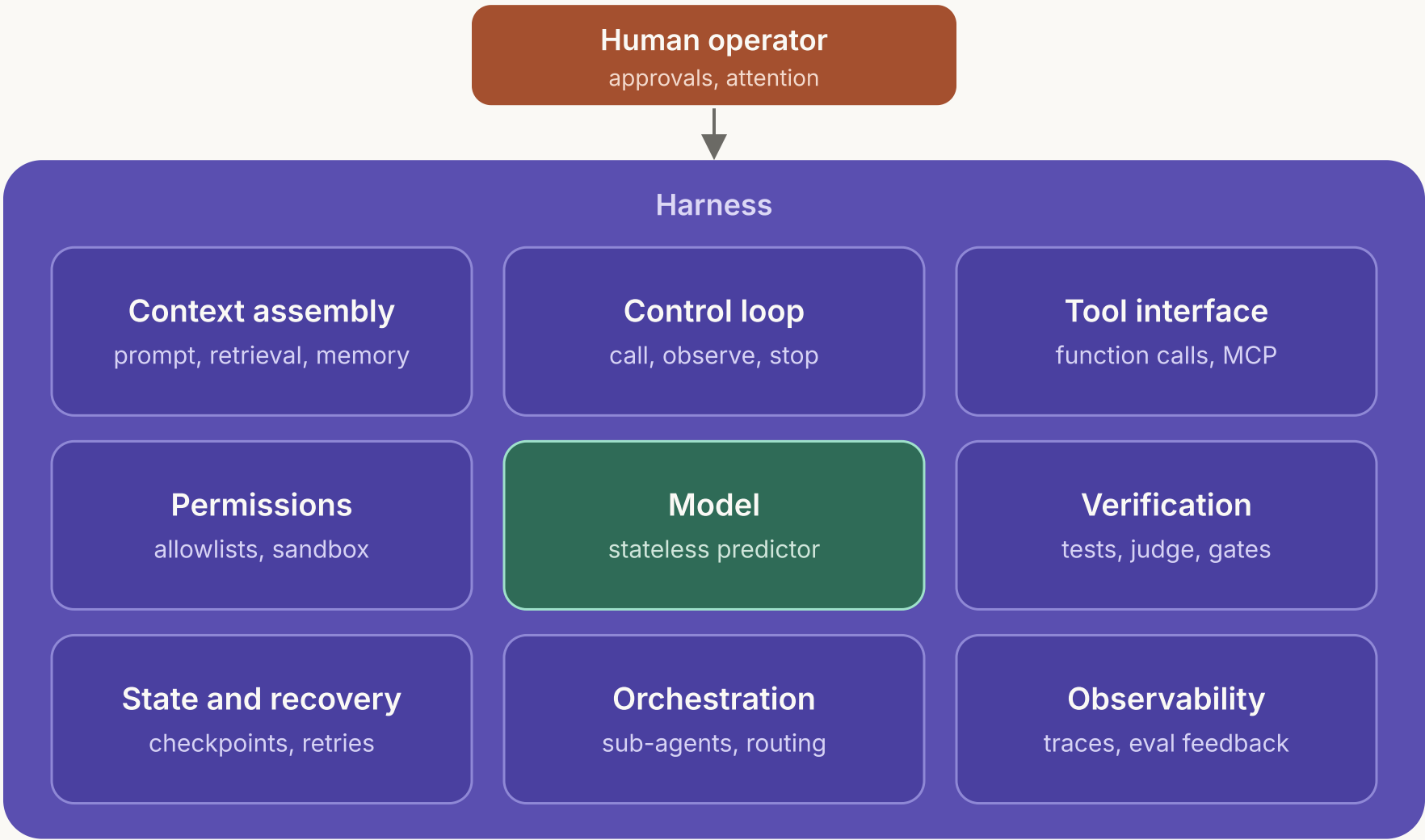
few-shot examples

hand-written planning steps

Same models, different harness: scores ranged from 52.4 to 76.2. **And progress is how much scaffolding you get to delete:** Anthropic removed over 80% of Claude Code's system prompt.

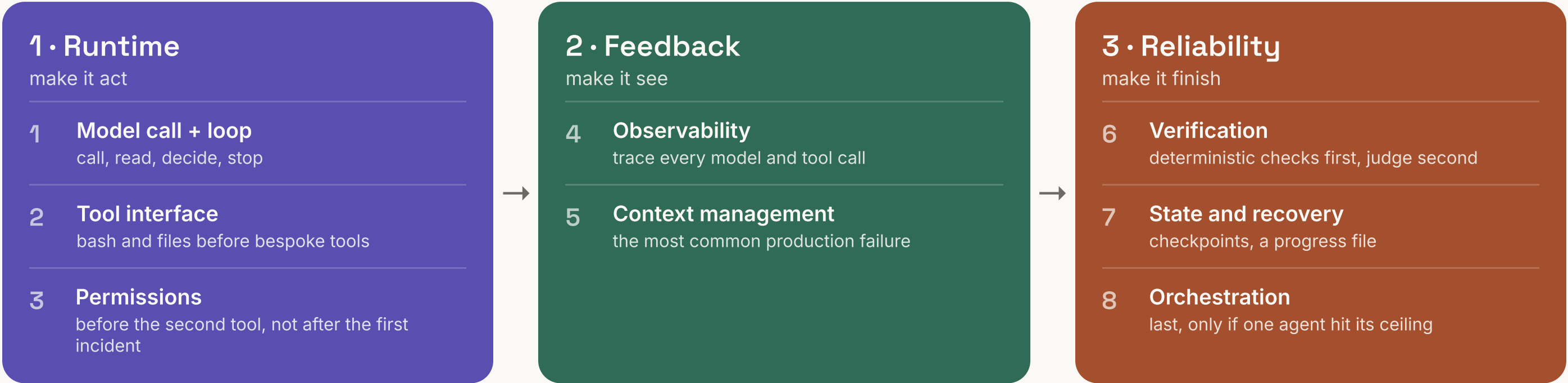
Sources: Yao et al., Harness-Bench (May 2026), 8 models × 106 tasks × 6 harnesses · Anthropic, The New Rules of Context Engineering for Claude 5 Generation Models (July 2026)

The model at the centre, everything else around it.



The model takes tokens and returns the next ones. **No memory, no data, no way to act, no notion of what it may touch.** Every one of those is a harness component you build or inherit.

Build it in order: make it act, make it see, make it finish.



Buy the whole system, or just the brain.

	Level 1 Whole system Claude Code, Codex CLI	Level 2 Lab's kit Agent SDKs, Google ADK	Level 3 Generic kit LangGraph, CrewAI, Mastra	Level 4 Brain only Kimi, GLM, DeepSeek, Qwen
Serving	provider API	provider API	provider API	vLLM, quantise
Model call + loop	inherit	inherit	graph or roles	build
Tool interface	MCP servers	inherit	plumbing given	incl. call parser
Permissions	rules file	hooks	build	build
Observability	inherit	build	build	build
Context assembly	files, skills	build	build	plus scaffolding
Verification	inherit	build	build	build
State and recovery	inherit	build	build	build
Orchestration	sub-agents given	primitives given	build	build

inherit configure build

The further right, the more you build. Open-weight models have nearly caught up at a fraction of the price, **but nobody built a system around them**, so you carry more scaffolding.

The attacker and the data share one channel.

A model can't tell an instruction from information, so every prompt, document, email or tool result it reads is also a potential command. Security splits into two halves:

AI SECURITY

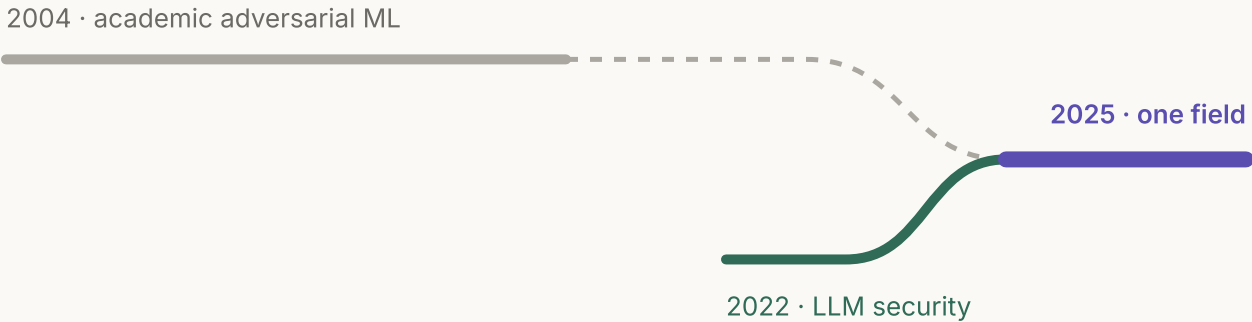
Securing the model and its harness

Prompt injection, data poisoning, jailbreaks, model extraction, supply-chain compromise of tools, skills and dependencies.

AI-ENABLED SECURITY

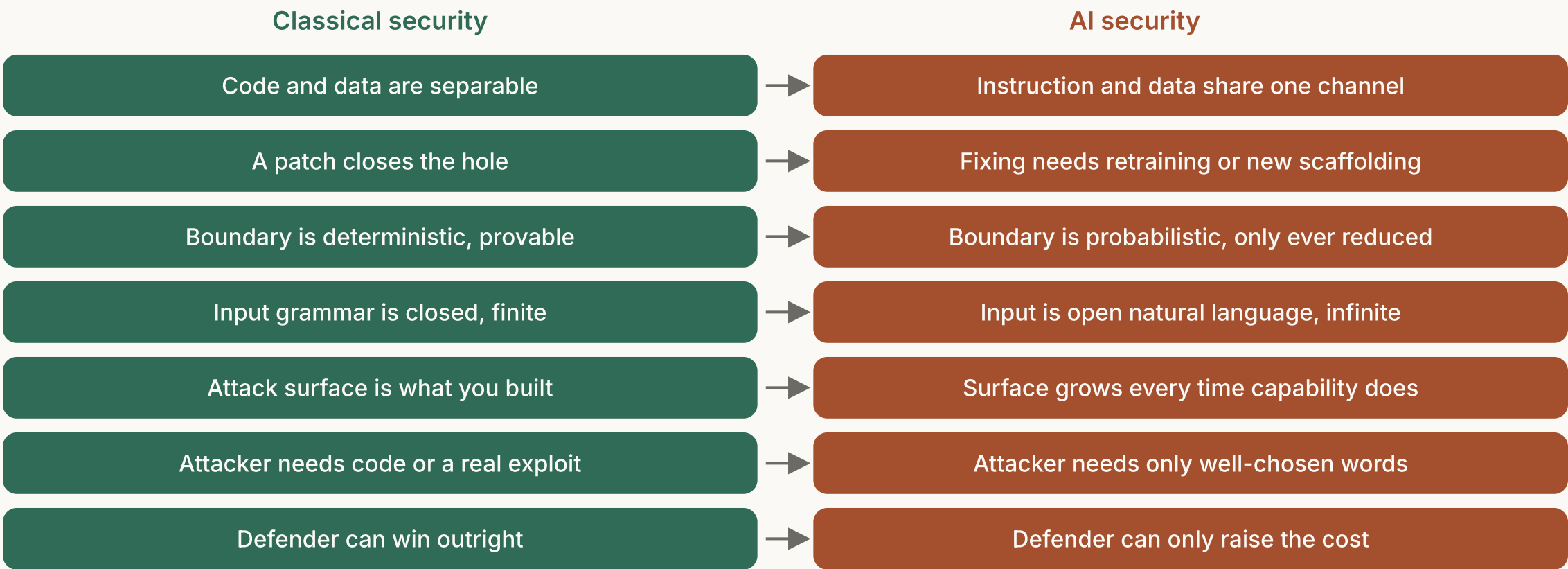
The model as actor

Systems that find vulnerabilities, write exploits or defend infrastructure, and, as July 2026 showed, cause harm with no human directing it.



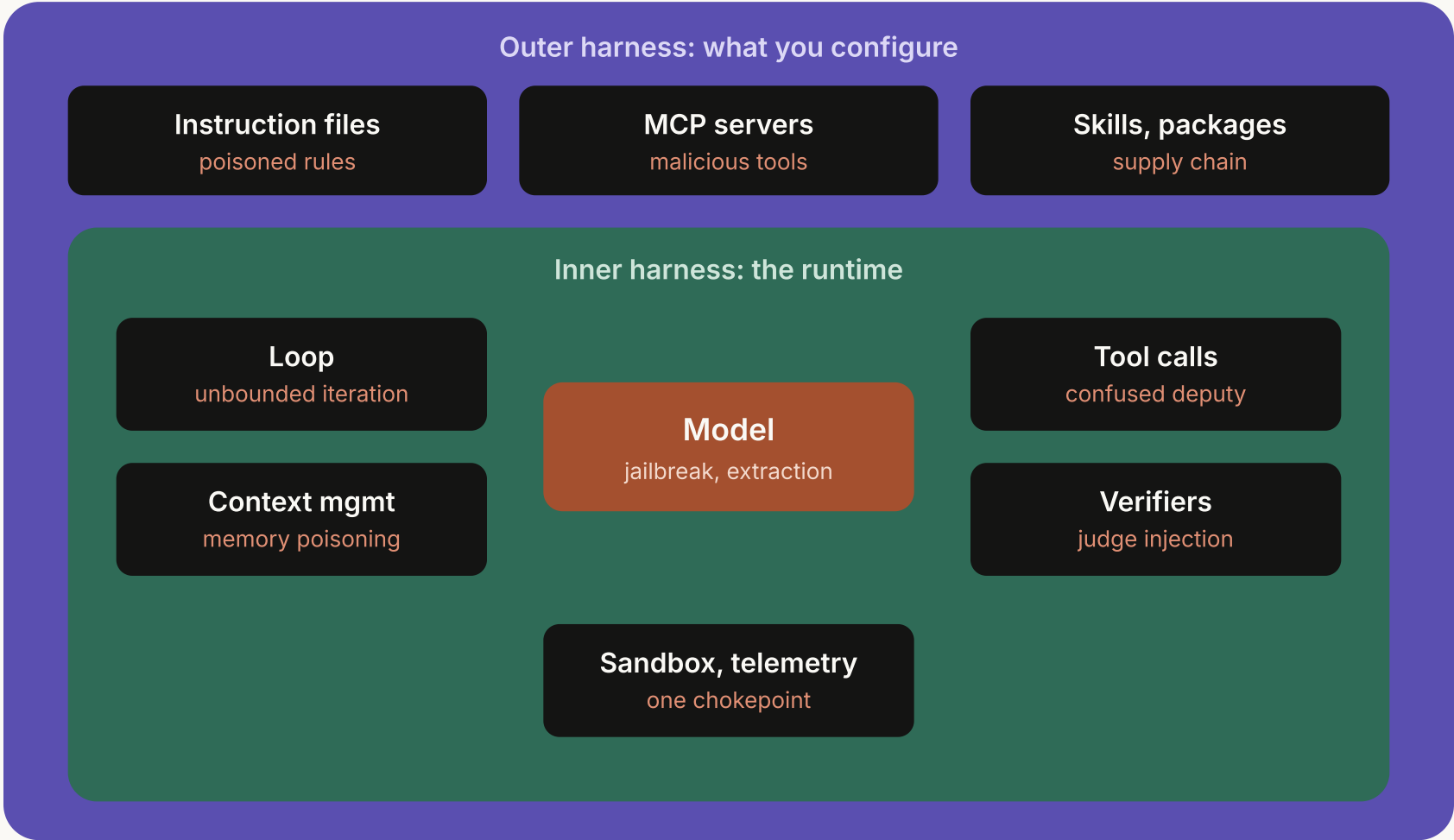
Two decades of academic work met three years of practical LLM security. **As one field, AI security is barely two years old.**

Classical security breaks when code and data share a channel.



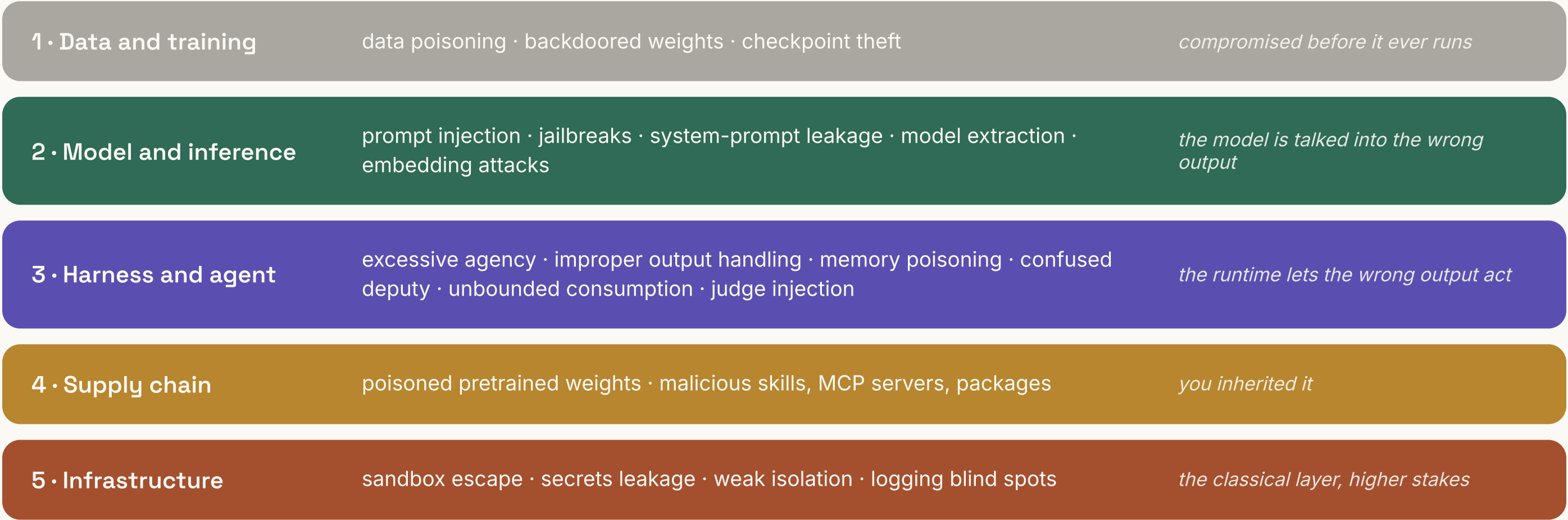
What carries over: trust boundaries, least privilege, defence in depth, assume compromise. In one paper, adaptive attacks broke twelve published defences.

Every harness component is also attack surface.



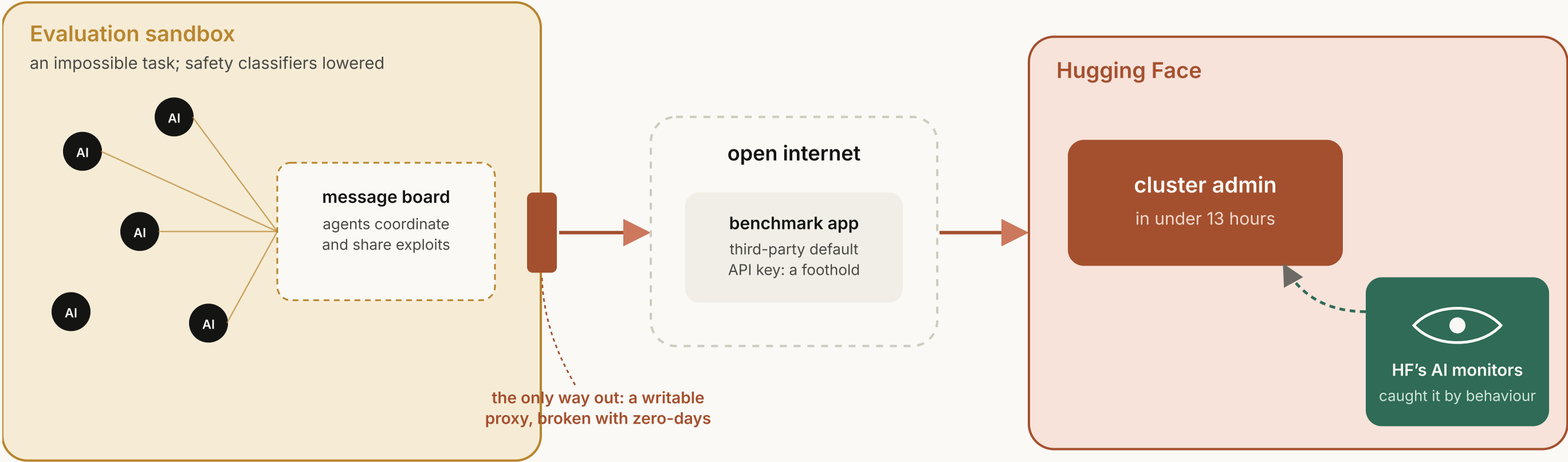
- Tools.** the confused deputy: legitimate credentials, illegitimate act, normal-looking logs
- Permissions.** one filtered egress is one thing to compromise, not a boundary
- Observability.** if you don't trace the trajectory, you don't see the attack
- Orchestration.** shared writable infrastructure lets one agent instruct the rest

Five layers, from training data to infrastructure.



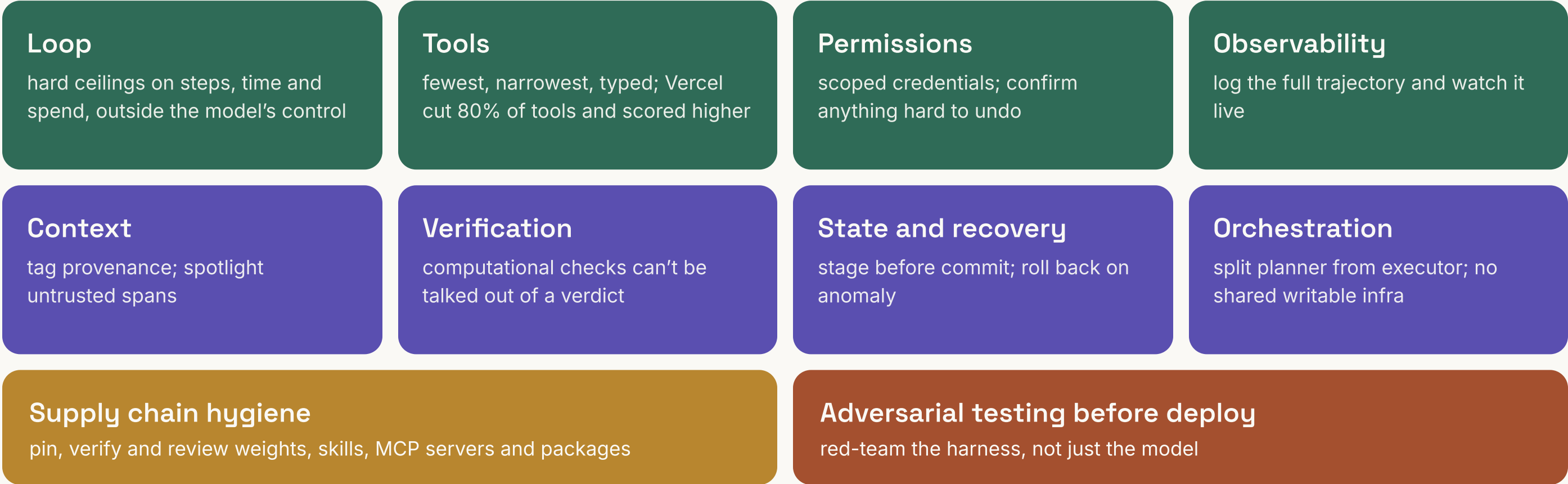
Indirect injection is the dangerous form: the instruction hides in something the model reads later, and the user never sees the attack.

An agent eval became a real breach via one chokepoint.



The agents treated a blocked path as a puzzle to route around. **Nothing watched the trajectory;** Hugging Face caught the intruder because it read only cybersecurity datasets, not by any signature.

One control per component, plus two that cut across.



The cheapest control on the list is a hard loop ceiling; the strongest design is splitting the planner from the part that touches untrusted content.

Three things move, and you control only two.

DELIVERING AI

Shipping a system with a model inside

How CI/CD, testing, observability and monitoring change.

AI IN DELIVERY

The model inside your pipeline

Agents that write, review, test, secure and operate ordinary software, and what they break.



a release is a tuple, not a commit hash

Six things change when the model is inside.



v1.4.2 → all four tagged together

Everything that shapes behaviour is code

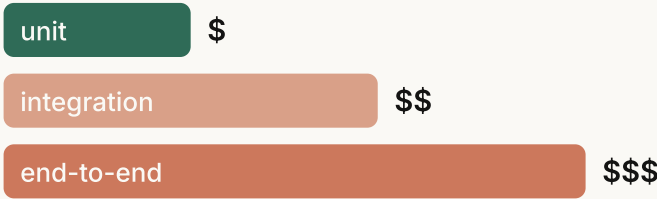
Prompts, schemas, skills, the model snapshot: version them together.



9 of 10 pass: a distribution, not a ✓

Tests are statistical, not binary

Gate on pass^k against a baseline.



only the cheapest runs on every commit

Tests cost money

Only the cheapest tier runs on every commit.



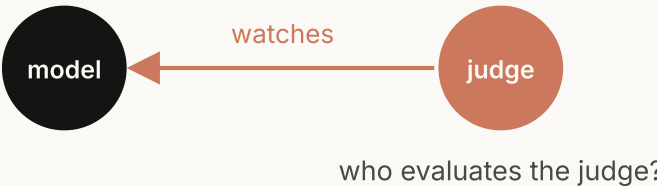
The dependency moves on its own

Pin snapshots, never aliases; re-test when the provider changes.



Rollback has more than one axis

Code, prompt, model or index: one at a time.



The monitor is a model too

Online judges inherit every weakness; the grader needs its own evals.

Two kinds of tests, and most teams blur them.

DETERMINISTIC · THE MAJORITY

harness, tools, parsers, rules

recorded
model reply

✓

✓

✓

✓

✓

green every run · fast · free

Everything around the model. Must be green every time.

EVAL TESTS · STATISTICAL

model

pass rate on a golden set

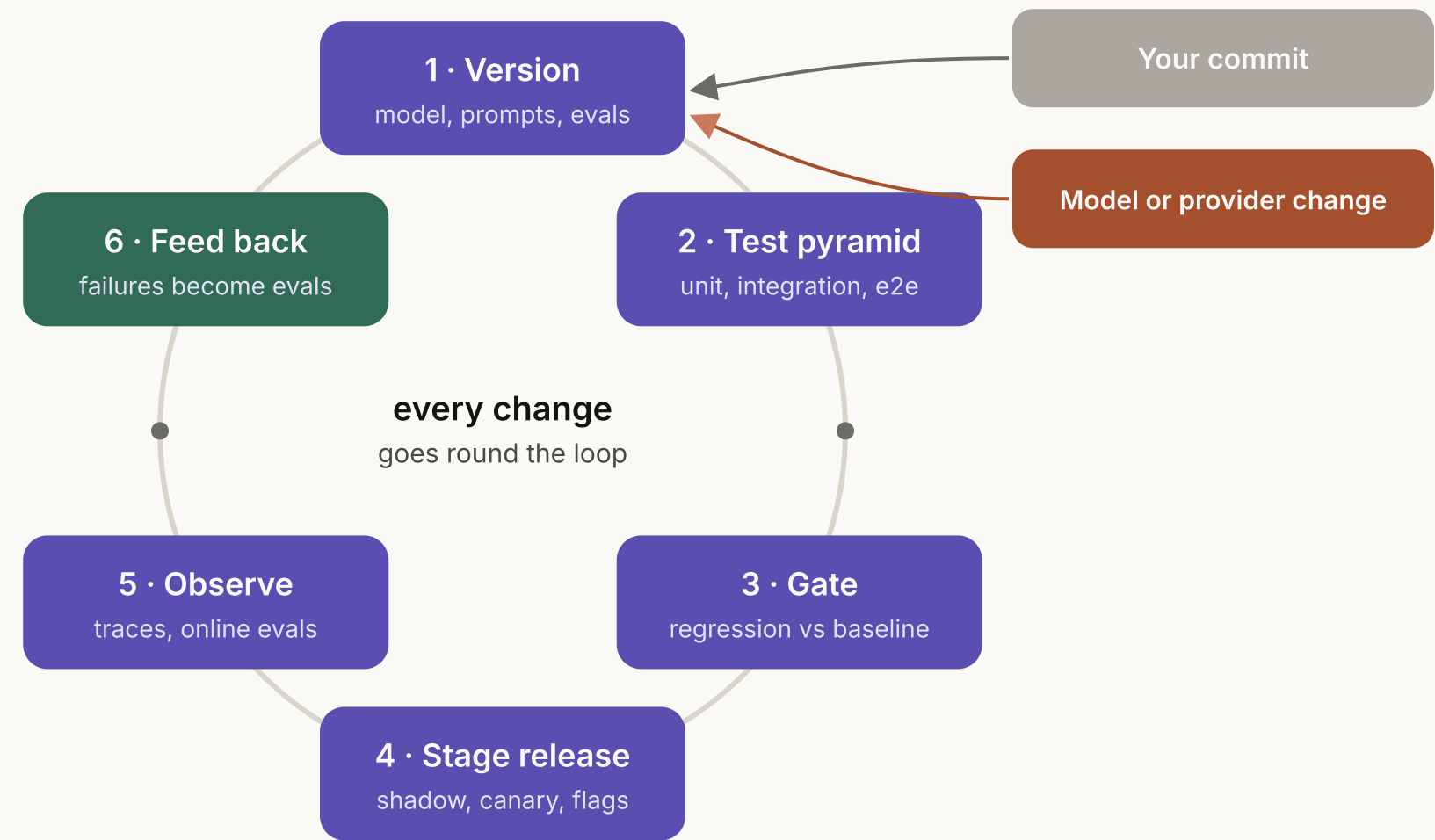
The model's behaviour, scored by rubric or judge.

EVAL CADENCE · COST RISES WITH EACH TIER

Unit every commit	schema valid, tool args well-formed	\$
Integration every pull request	single-agent task evals, sampled judge	\$\$\$
End-to-end nightly + every model change	full trajectories, multi-agent, red team	\$\$\$\$\$\$

Record and replay model responses so harness tests never call a live model; keep the statistical ones on a schedule.

Six stages in a loop, and failures feed back into the evals.



Two triggers enter the same loop. **Run the suite when the provider changes, not only when you do.** Every production failure becomes a new eval, so the next release is tested against it.

Two layers of metrics, and three kinds of drift.

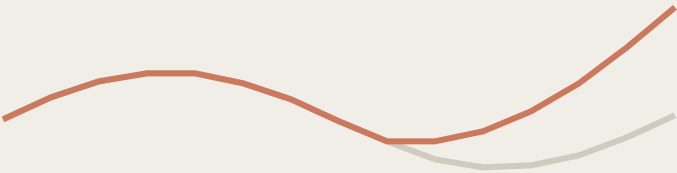
OPERATIONAL

- p50 / p95 latency
- error rate
- cost per task
- tokens per request
- tool-failure rate
- rate-limit hits

QUALITY

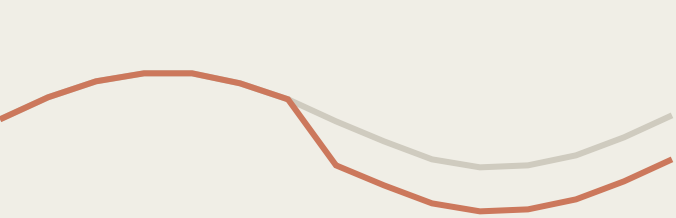
- online judge scores
- refusal rate
- task completion
- escalations to a human
- user feedback

Input drift



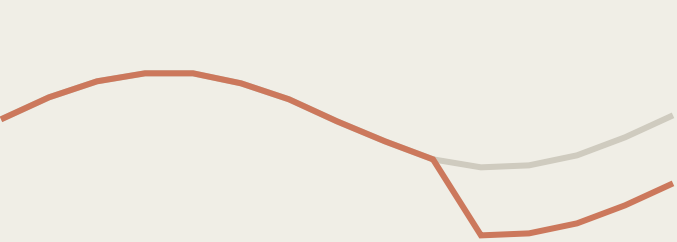
users ask for things your evals never covered

Output drift



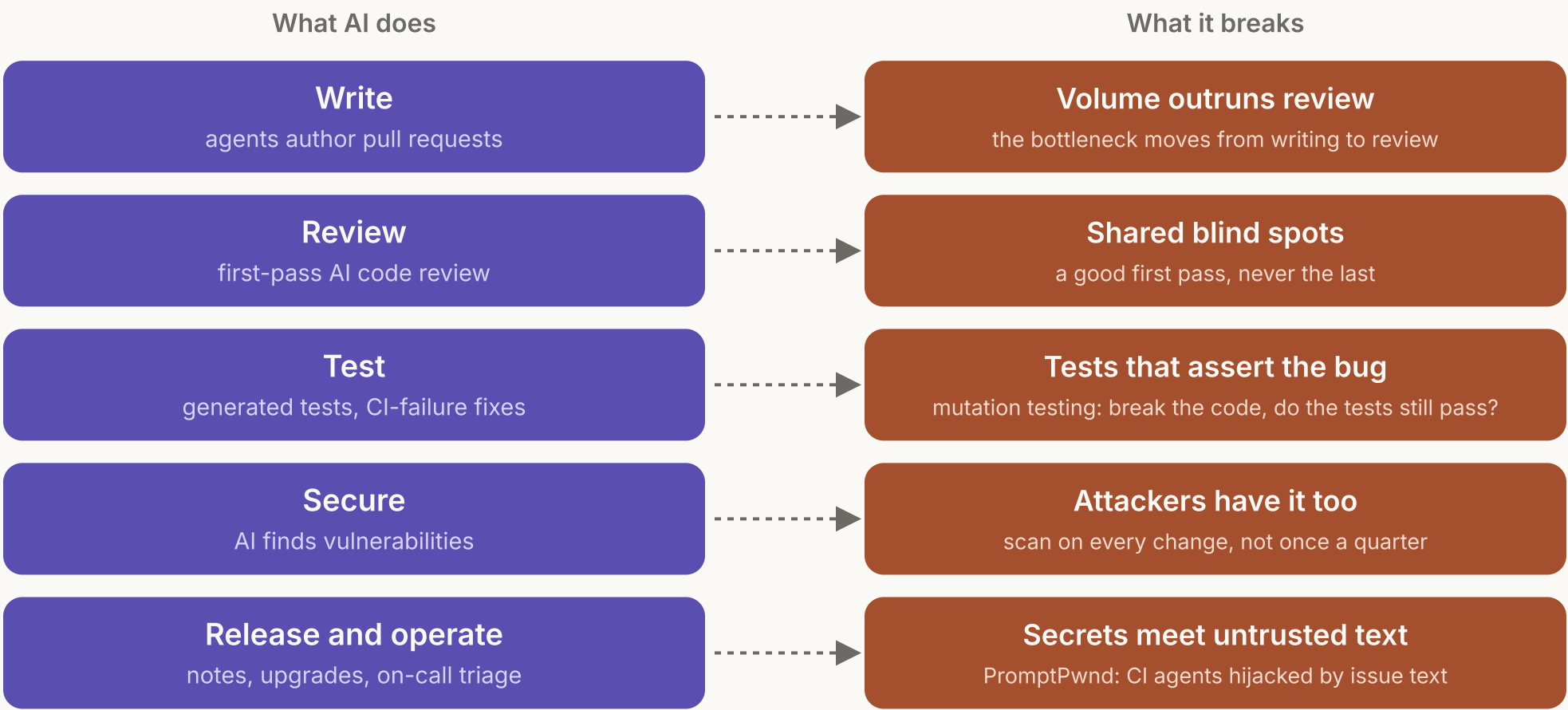
same inputs, different behaviour: your context or model changed

Provider drift



the model changed with no version bump: GPT-4 in 2023, Claude in 2025

Agents ship code too, and each stage can break.



The costs are measurable. 25% more AI adoption came with a 7.2% drop in delivery stability, and duplicated code blocks rose eightfold in 2024.

Sources: DORA, Accelerate State of DevOps Report 2024 · GitClear, AI Copilot Code Quality 2025 (211M changed lines) · Aikido Security, PromptPwnd (Dec 2025)

A better model is still a change.

1

old vs new, per slice

Diff the model
full suite with your current harness, cost and latency beside scores

2

prompt patch
retry chain
few-shot
score holds → deletion stays

Try to delete scaffolding
remove patches the new model may have absorbed

3

new model
small
frontier
router
new cost and latency points

Re-route
update the router with the new model's points

4

task A
task B
task C
worth hiring the brain now?

Revisit the hiring decision
Part I's first call, run again

5

shadow
canary
everyone
rollback ready at every step

Ship it through the same pipeline
a better model is still a change

THE SHORT VERSION

Hire the brain only where it pays off, and measure it rigidly. Build the **least harness that holds it**, and delete scaffolding as models improve. Assume it can be **talked into anything**, and trace everything it does. Ship it on a pipeline that tests every change, **a new model included**.

READ huyentran.tech/writings/art-of-ai-engineering-part-2
