

Keeping AI Agents Honest

What AI Platform Engineers and Architects Actually Do, and How to Do It

Bill McCoy

Engineering, not generating.

Bill McCoy

Written by AI agents. Kept honest by the harness Bill McCoy built, the one this book teaches.

Contents

About this book	4
Where AI Stands, and Where This Job Sits	10
PART I. GETTING STARTED	
1 What an AI Platform Engineer Does	17
2 AI Models, Just Enough	31
3 What Makes an Agent	43
4 Anatomy of a Harness	54
5 Setting Up the Companion Repo	63
PART II. CONTEXT: WHAT THE AGENT KNOWS	
6 Instruction Files That Stay Small	75
7 The Repository as Source of Truth	84
8 Keeping Documentation Honest	93
9 Retrieval Without the Hype	103
10 Long-Running Work Across Sessions	118
PART III. TOOLS: WHAT THE AGENT CAN DO	
11 Designing Tools for Agents	127
12 Building an MCP Server	141
13 Governing MCP Servers Across an Organization	155
14 Multi-Agent Patterns	174
PART IV. GUARDRAILS: WHAT THE AGENT CAN'T BREAK	
15 Fitness Functions and Structural Tests	188
16 Boundary Rules	198
17 Custom Linters That Teach	211
18 Policy Engines	221
19 Permissions, Sandboxes and Human Approval	232
20 The Agent Attack Surface	250
PART V. EVALUATION: KNOWING WHETHER IT WORKS	
21 Evaluating Output That Changes Every Run	267
22 Model Judges	281
23 Evaluations as a Gate	296
PART VI. FEEDBACK LOOPS	
24 What Belongs in CI When Agents Write the Code	311
25 Closing the Loop	322
PART VII. RUNNING IT	
26 Measuring a Harness	333
27 Gateways, Cost and Model Routing	348
28 The Central AI Team	366

29	Rolling It Out to Teams	382
30	Zero to Prod: One Path from a Request to Production	397
31	Walking Into a Harness Someone Else Built	414

PART VIII. YOUR CAREER

32	Skills, Learning Path and Portfolio	423
33	Interviewing for AI Platform Roles	435

PART IX. QUICK LISTS

34	Ten Failure Modes and How to Catch Them	447
35	Ten Guardrails Worth Adding First	457
36	Ten Questions to Ask in Your First Week	466

APPENDICES

A	The Same Harness in Five Tools	475
B	The Zero to Prod Kit	485
C	A 90-Day Plan for a New AI Platform Role	513
D	Dogfooding: How This Book Was Built	524
E	Glossary	534
F	Sources	559
G	Quick-Reference Card	588

About this book

Copyright and license

Copyright 2026 Bill McCoy. All rights reserved. Published by GraceZero.

No part of this book may be copied, shared or published in any form or by any means without written permission from the author, except for brief quotations in reviews and articles.

The companion repository's code and templates are under the MIT License, in its `LICENSE` file, so you may use, change and share them in your own work.

What this book is

This is a hands-on guide for engineers and architects who have to make AI agents safe, reliable, measurable and affordable inside an organization. It covers two kinds of agent:

- **Coding agents**, which work in your codebase: Claude Code, Codex CLI, GitHub Copilot, Cursor, Devin Desktop (formerly Windsurf) and others.
- **Product agents**, which work inside your application: an assistant that reads records, calls tools and takes actions for your users.

Its subject is everything around the model: what the agent is told, what it's allowed to touch, the checks its work has to pass, how you know whether it's any good, and what it costs. The book calls that the **harness**. The model usually comes from someone else; the harness is yours to build. Most of the job postings behind chapter 1 ask for work on agents, and two-thirds or more ask for observability, evaluation, guardrails and security. All four are harness work.

Every guardrail in the book is built and run in a companion repository: a small helpdesk application with an agent inside it. You don't take a check on trust. You run it, break it on purpose, and watch it fail with a message that says what to do.

By the end, you should be able to:

- Read an "AI platform engineer" or "AI architect" job posting and say which of five different jobs it describes.
- Explain what an agent is, what a model call costs and why its output changes from run to run, well enough to predict how it fails.
- Give an agent the context it needs without drowning it, and keep what it reads true with checks.
- Design tools and MCP (Model Context Protocol) servers that act with the permissions of the person the agent works for.

- Turn rules your team keeps in prose into checks that fail the build with a message an agent can act on, and put a person's approval in front of the actions that matter.
- Tell whether an agent is any good when its output changes every run, and hold back a new prompt, model or tool until it passes.
- Run it all for an organization, with one gateway for model calls, budgets, and numbers that say whether the harness works.
- Take a request to production along one path, with a person deciding at each gate and a page at the end that says which claims were measured to hold.
- Walk into a harness someone else built and choose the first thing to fix.

Who it's for

Engineers and architects who already know software delivery: git, the command line, APIs, CI and containers. You're new to AI agents, or you've used them yourself and now have to make them safe for other people. Most often that means a platform, DevOps (development and operations), infrastructure, site reliability or backend engineer moving into AI platform work; an architect who sets the patterns and gates those teams build on; or a tech lead who has to hire for either. The introduction describes both roles, and the jobs around them.

The book assumes nothing about AI. Every AI idea is explained from first principles, starting with the smallest real example, and every term is defined the first time it appears.

It's probably not the book for you if:

- **You want to train, fine-tune or serve models, or run GPU (graphics processing unit) clusters.** That's the infrastructure kind of AI platform job. Chapter 1 shows how to spot it in a posting and points to books that cover it.
- **You're new to software delivery itself.** The first chapter's lab already assumes a terminal, git and Node.js.
- **You want a reference for specialists, or a tour of every product.** This is plain English for engineers and architects new to the field. It goes deep on a few mechanisms, and dates what it says about any product, because product names and features change.
- **You want a lab built on a coding tool other than Claude Code, or on a model API other than Anthropic's or OpenAI's.** The companion repository's hooks and settings are Claude Code's, and its clients for a real model call Anthropic's API and OpenAI's; everything else runs on a mock. Appendix A maps the same harness across five coding tools, and the chapters teach each general idea with more than one vendor's version.

What it covers, and what it doesn't

The book opens with an introduction, then has nine parts and seven appendices.

Part	Chapters	What it covers
Introduction	Before chapter 1	The roles, where AI stands today, who makes the models and how they reach you, and what changes fast
I. Getting Started	1 to 5	The job family, from a dated sample of real postings; AI models, just enough to reason about them; what makes an agent; the eight parts of a harness; setting up the companion repository
II. Context: What the Agent Knows	6 to 10	Instruction files that stay small; the repository as the source of truth; documentation checked against the code; retrieval without the hype; work that spans many sessions
III. Tools: What the Agent Can Do	11 to 14	Designing tools for agents; building an MCP server; governing MCP servers across an organization; multi-agent patterns
IV. Guardrails: What the Agent Can't Break	15 to 20	Fitness functions and structural tests; boundary rules; custom linters that teach; policy engines; permissions, sandboxes and human approval; the agent attack surface
V. Evaluation: Knowing Whether It Works	21 to 23	Evaluating output that changes every run; model judges; evaluations as a gate
VI. Feedback Loops	24 and 25	What belongs in CI when agents write the code; sending a failed check back to the agent
VII. Running It	26 to 31	Measuring a harness; gateways, cost and model routing; the central AI team; rolling it out to teams; Zero to Prod, one path from a request to production; walking into a harness someone else built
VIII. Your Career	32 and 33	Skills by kind of job, a path through the book and a portfolio; interviewing for AI platform roles
IX. Quick Lists	34 to 36	Ten failure modes and how to catch them; ten guardrails worth adding first; ten questions to ask in your first week

The appendices:

- **A. The Same Harness in Five Tools.** Where Claude Code, Codex CLI, GitHub Copilot, Cursor and Devin Desktop read their instructions, where their hooks live, how they connect MCP servers, and how each runs without a person at the keyboard. Every entry is dated.
- **B. The Zero to Prod Kit.** Every file chapter 30's path produces, from the intake brief to the proof page, 40 files in the companion repository's `templates/` folder, in the order its stages produce them. Each comes as a skeleton to copy, a filled example and the check that proves it, and all but the week-one assessment checklist, which is for a person, are proved by a test in the companion repository.
- **C. A 90-Day Plan for a New AI Platform Role,** in four phases, with a 60-day cut for a short engagement.
- **D. Dogfooding: How This Book Was Built.** The book's own harness, from its records, and what it never measured.
- **E. Glossary.** The terms the book defines, in plain words, each with the chapter that explains it.
- **F. Sources.** Every work and document the book cites, what each is good for, the chapters that cite it and the date it was read.

- **G. Quick-Reference Card.** One page: the eight parts of a harness, the lab's main commands, the first ten guardrails, week one and its ten questions, and the ten failure modes, each pointing to its chapter.

How to use it

Read the introduction, then Part I, because the rest of the book uses their vocabulary. After that, every chapter stands on its own, in any order.

Every chapter opens with three bullets that say what the chapter gives you. If none of the three is something you need today, skip it. Chapters 1 to 33 then take the same steps: the problem, how it goes wrong, how it works, a Try it, a field note where there is one, how to tell it's working, what to check before you rely on it, how to talk about it in an interview, and a short Keep this list. The three quick lists in Part IX, chapters 34 to 36, keep the same frame, with ten items in place of the problem, how it goes wrong and how it works.

Two more labels keep the book honest. A made-up example is marked "illustrative scenario," and anything real comes from a cited source or a field note. Each "Check before you rely on it" list names the parts of its chapter most likely to go stale, with dates.

Paths through the book

If you're	Read, after Part I
Heading for an enablement job: a central team whose AI platform other teams build on	Part VII, then Parts IV and V
Building the harness around coding agents	Parts II, IV and VI
Building agents into a product	Parts III and V, then chapter 19
Running one AI system for one purpose (what chapter 1 calls applied)	Parts III and V
Setting up how requests become production systems	Chapter 30, then Appendix B
Starting a new role, or inheriting a harness	Chapters 31 and 36 in your first week, then chapter 35 and Appendix C
Leading a team, or hiring for one	Chapter 1 (it also says how to write a posting), Part IX, then chapters 26, 28 and 29
Preparing for interviews	Chapters 32 and 33, and the interview section at the end of each chapter you read
Short on time	Part IX: three lists of ten, each pointing to the chapter that goes deeper

The first four rows follow chapter 1's kinds of job. Chapter 32 turns each into an order through the whole book, and a script in the companion repository, `postings/skills.mjs`, prints that order for any kind of job. The last five rows are my suggestions.

Try it, and what it costs

Every chapter, and Appendices B and C, has a Try it section: steps to run in the companion repository, with the exact commands and the output to expect. Most include a step that breaks something on purpose, so you see a guard fail before you rely on it. Each chapter has a tag in the repository, and every Try it starts by checking out its tag, which gives you the repository at the commit that chapter's Try it was checked against, so your output matches the book's. A tag can hold more than its chapter builds: chapter 9's tag already contains the code chapters 15 to 18 and 24 explain, and several tags share a commit (9 and 10, 14 to 18, 24 and 25). Chapter 5 explains the tags and the setup that goes with them.

Everything runs against a mock model by default, at no cost. The mock plays the model's part from a fixed script, so it needs no network connection or account and gives the same output every time. It checks the machinery, not a model's judgment; chapters 21, 22, 23, 26 and 27 each say what only a real model can answer. Spending money is something you ask for, one command at a time. A command calls a real model only when you add `--real` and the credential for its model's provider, Anthropic or OpenAI, is present, and every such run is bounded: the evaluation commands refuse `--real` without a dollar cap, the triage assistant stops at its turn limit, and every real call passes the lab's budget check first. Treat `--real` itself as the switch that spends money.

The callouts

- **Field note:** something that happened in my own production systems, anonymized, with real numbers, failures included, and checked against those systems' records. They are one engineer's small portfolio of apps, so they show how a mechanism behaves, not how it scales.
- **Try it:** steps to run in the companion repository, with the output to expect.
- **Watch out:** a mistake that costs you.
- **Under the hood:** detail you can skip on a first read.
- **Keep this:** the lines to remember. Every chapter ends with a few.

The companion repository

The companion repository, `gracezero-agent-harness-lab` on GitHub, holds everything the labs run: a small helpdesk application with a triage assistant inside it, a deterministic mock model with clients for Anthropic's API and OpenAI's behind the same interface, the book's guardrails with tests that plant a violation to prove each one catches it, chapter 1's job-postings sample, and Appendix B's templates. It was written from scratch for this book and contains none of my private code or data. Chapter 5 walks through all of it.

What you need: Git 2.45 or newer (`git --version` shows yours), Node.js 24 or newer, and Python 3.12 or newer, plus an internet connection the first time setup runs. The repository's CI runs it with Python 3.14 on Linux, Windows and macOS, and with Python 3.12 on Linux. Chapter 16's Go, Java and .NET rules run only in CI, so you don't need those languages.

The repository's address: <https://github.com/BillM9000/gracezero-agent-harness-lab>

The book's page: <https://gracezero.com/keeping-ai-agents-honest>.

Getting started. On macOS or Linux:

```
git clone https://github.com/BillM9000/gracezero-agent-harness-lab ~/src/agent-harness-lab
cd ~/src/agent-harness-lab
node setup.mjs
node check.mjs
```

On Windows, clone into a folder with a short path, such as `c:\src` (chapter 5 explains why):

```
git clone https://github.com/BillM9000/gracezero-agent-harness-lab C:\src\agent-harness-lab
cd C:\src\agent-harness-lab
node setup.mjs
node check.mjs
```

`node setup.mjs` installs every package at the exact version the lock files list, and is safe to run again. `node check.mjs` runs every check the repository has, one line each, apart from chapter 16's Go, Java and .NET rules, which CI runs in jobs of its own, and chapter 24's mutation run, which takes minutes and runs nightly. It's the command CI runs.

How this book was built

This book was built inside the harness it teaches: a spec, a short instruction file, a checker that fails when a citation doesn't resolve, a fresh copy of the repository for every Try it, and a person approving whatever can't be undone. The line under the epigraph says who did what. Appendix D, "Dogfooding: How This Book Was Built," is the long version, told from the records.

Where AI Stands, and Where This Job Sits

- Which job you're looking at: the architect, the platform engineer, and the roles around them.
- Where AI (artificial intelligence) stands today, in the words the rest of the book uses.
- Who makes the models, how they reach your company, and what stays true when the names change.

The roles, right away

If you came here from a job posting, the first question is which job it describes. Two titles cover the work in this book. Several nearby titles cover work it leaves out. This section sorts them, and then walks through a working week in each kind of job.

What an AI architect does

An AI architect decides how a company builds with AI, and writes it down so that other teams can follow it. The work comes in three forms: patterns, standards and gates. A pattern is a tried way of building something, such as how an assistant finds the documents it needs, how an agent is given a tool, or where its logs go. A standard is the version every team has to meet. A gate is the review a project passes before it ships, where someone with the authority to say no looks at the evaluation results, the data the system touches and the plan for when it fails.

The architect also makes the build-versus-buy calls. Do you buy model access from the company that makes the model, or through the cloud account your company already has? Do you run an open model on your own hardware? Do you buy a coding agent for every engineer, and build the checks around it yourselves? Each answer carries a cost, a risk and a degree of lock-in, and the architect has to be able to say which.

Then there's governance: who owns the rules for AI, which uses need a second person's approval, and what gets logged and for how long. And there's the room with executives in it. Leaders, lawyers and security staff decide how much risk the company will take with AI. The architect explains the trade-offs in their terms, then turns their decision into a standard a team can build against.

What an AI platform engineer does

An AI platform engineer builds what the architect describes, and proves that it works. This book calls most of it the harness: everything around the model that steers an agent before it acts and checks its work afterward. The engineer builds the harness, breaks it on purpose to see each check catch what it should, and measures whether it helps.

In practice that starts with a gateway, the one place every model call in the company goes through, which holds the keys, enforces budgets and records who spent what. It means checks: tests, lint rules and policies that fail a change with a message an agent can act on, and a person's approval in front of the actions that matter. And it means evaluations: sets of cases, scored the same way every time, that decide whether a new prompt, model or tool may ship. The engineer owns the numbers, too: how often each check fires, what each team spends, and how often an agent's work has to be redone.

One body of work, two titles

The two descriptions overlap because they often describe the same work from two distances. Which title a company uses varies, and a posting rarely says why. Some call it a staff or principal platform engineer, and others an architect: AI architect, AI solutions architect, enterprise AI architect. In chapter 1's sample, architect postings ask for more seniority and cover the same topics, from agents and MCP (Model Context Protocol) to evaluation and observability, with different weight: more often retrieval, guardrails and a person's approval of what agents do, and less often cost control.

This book is for both. Some roles are both at once: an architect's job done with an engineer's hands, where the person who sets a standard also builds the check that enforces it. If that's your job, you'll use every part of the book.

The neighbors

Several nearby jobs share words with these two. Each is real work, and this book covers little of it.

- **Data scientist.** Analyzes data and builds statistical and machine learning models to answer questions for the business. A data scientist may use the platform you build, but rarely runs it.
- **Machine learning engineer.** Builds, trains and ships machine learning models as part of a product. For them the model is the output; in this book the model is an input.
- **MLOps (machine learning operations) engineer, and the infrastructure kind of AI platform engineer.** Runs the pipelines that train, deploy and monitor models, and the GPU (graphics processing unit) clusters and serving systems underneath them. Many of these postings carry the same title, AI platform engineer, as the work this book teaches. It's the one job with that title the book leaves out, and chapter 1 shows how to spot it in a posting.
- **AI engineer.** Builds applications on top of models: an assistant, a summarizer, a search feature. The platform in this book is what an AI engineer builds on, and product agents sit close to their work.
- **Prompt engineer.** Writes and tests the instructions and examples a model is given. Every role here does some of this. The book treats a prompt like code: kept in a file, reviewed, and tested before it ships.

- **Solutions engineer and forward-deployed engineer.** Both work with customers. A solutions engineer helps a customer fit a product to their systems, often as part of a sale. A forward-deployed engineer is placed with a customer to build on the vendor's platform. Both use many of this book's skills, on someone else's systems.

A working week in each kind of job

Chapter 1 sorts the jobs behind these titles into five kinds, using a dated sample of real postings. Here is a Tuesday from a working week in each. Each is an illustrative scenario, pieced together from what those postings describe; none is one person's diary.

Enablement. You're on a central team whose AI platform other teams build on. The morning starts in the gateway's spending report: one team's cost has doubled in a week. You trace it to an assistant that retries every failed call, and you set a limit. At eleven you run office hours for two teams building their first agents. After lunch a team brings its design to the review gate. You ask how they'll know the agent works, point them to the shared evaluation tooling, and book them in again for next week. You end the day sorting new requests from people outside engineering who want to build their own tools.

Coding agents. You own the harness around the agents that write your company's code. A pull request from a coding agent has failed CI (continuous integration) three times on the same lint rule, so you rewrite the rule's message to say what to change, and the agent's next attempt passes. You review a proposed addition to the repository's instruction file and cut it to two lines. You connect an internal MCP server to the agents with read-only access. Before you leave, you pull the week's numbers: cycle time, time waiting for review, and how often changes written by agents are reverted.

Product agents. You build the platform under an assistant that customers use inside your product. It proposes replies and actions, and a person approves the ones that matter. Today a new prompt is waiting to be promoted. You run it against the evaluation set: accuracy holds, but the cost of a conversation goes up, so you take both numbers to the product lead. In the afternoon you look at the approval queue. One kind of action is approved within seconds every time, a sign that nobody reads it, so you open a ticket to change how it's shown.

Applied. You run one AI system for one purpose, and other people depend on it: a code security scanner built on an LLM (large language model), say. A team reports that it flagged safe code again. You add their example to the test cases, change the prompt, run the whole set, and ship only when the old cases still pass. Another week, Tuesday is a migration: moving a search system to a new agent framework while checking that its permissions, the quality of its results and its citations all survive the move.

Infrastructure. You keep the models themselves running. The morning is about GPU capacity: one cluster is full and a training job is waiting. In the afternoon you send a small share of traffic to a new model version, watch its latency and error rate, and widen the share when both hold. This is the job this book leaves out, and chapter 1 points to books that cover it.

And the architect. The title cuts across the first four kinds, and the architect postings in the same sample describe a Tuesday like this. A team brings an agent that answers customers' billing questions to the review gate. Its evaluation covers accuracy but not a customer who pastes instructions into a ticket, so you hold it until it has those cases, and point to the standard that requires them. At eleven you sit down with security and risk over what agents write to their logs: what a log may contain, who can read it and how long it's kept. You write the decision into the standard. In the afternoon a team asks whether to call a model through the company's cloud account or straight from the company that makes it. You write down the trade: the same model either way, but a different bill, contract and security review, and a list of features to check, because a channel can lack one the team uses. Then you send the calls through the gateway, so the choice can change later without touching their code.

Where things stand

How agents got here

First came chat. You typed a question, a model answered, and a person decided what to do with the answer. Then the model makers' APIs (application programming interfaces) added a way for a model to ask your code to run a function and send back the result. OpenAI and Google call this function calling; Anthropic calls it tool use. Put that exchange in a loop, let the model choose each next step until it reaches a goal or a limit, and you have an agent. Agents that read a repository, edit files and run the tests went to work in engineering teams. Agents that read records and take actions for users went into products.

This book's view is that once agents could act, the harder problem moved off the model and onto everything around it: what the agent is told, what it may touch, which checks its work has to pass, and how anyone knows whether it's good. The book calls that the harness, as Birgitta Böckeler's April 2026 article on martinfowler.com does, and chapter 4 takes one apart.

The words you'll meet

These are one-line versions. Chapters 2 to 4 take the model, the agent and the harness further, chapters 12 and 13 do the same for MCP, and Appendix E lists every term with the chapter that explains it.

- **AI:** the broad field of computer systems that do tasks once thought to need a person, such as understanding language. In this book it nearly always means the next term.
- **Generative AI:** AI that produces new text, code or images from a request, rather than only sorting or scoring what it's given.
- **LLM:** the kind of model this book is about, trained on large amounts of text to predict what comes next. Chapter 2 covers what that means for you.
- **Model:** the trained system you send a request to. A call to a model is an HTTP (Hypertext Transfer Protocol) request: your code sends JSON (JavaScript Object Notation) that holds the conversation, and gets JSON back.

- **Agent:** a model that calls tools in a loop until it reaches its goal or a stop condition. Chapter 3 takes one apart and runs it.
- **Harness:** everything around the model that steers the agent before it acts and checks its work afterward. Chapter 4 names its parts.
- **MCP:** an open standard for connecting AI applications to tools and data, so that one tool server can work with many agents. Chapters 12 and 13 build and govern servers.

WATCH OUT

Since 29 September 2026, an executive order, "Inaugurating The Era Of Super Intelligence," has United States federal agencies write "Super Intelligence" and "SI" where they wrote "Artificial Intelligence" and "AI" in documents that aren't law. The legal definition hasn't changed: the order defines the new terms by pointing to it. This book keeps "AI," as the field, the job postings and the people reading it do.

Who makes the models, and how they reach you

Picture three layers between your company and a model. Someone makes the model. It reaches your company through a channel. Your people use it through a tool. Each layer is a separate choice, and each has more than one answer.

Layer	What it is	Examples on 30 September 2026
Makers: frontier labs	Companies that train large general-purpose models and sell access to them	Anthropic (Claude models), OpenAI (GPT models), Google (Gemini models)
Makers: open weights	Companies that publish a model's weights, so you can run it on your own hardware or rent it from a host	Meta (Llama, Muse Glimmer), Mistral AI (Mistral Large 3), Google (Gemma), OpenAI (gpt-oss)
Channels: direct API	You buy from the maker: an account, a key, and a bill for what you use	Claude API, OpenAI API, Gemini API
Channels: cloud platforms	Models from several makers inside a cloud account your company already has, billed and governed there	Azure OpenAI in Microsoft Foundry, Amazon Bedrock, Gemini Enterprise Agent Platform (formerly Vertex AI)
Channels: gateways	One address every team calls, in front of every provider: it holds the keys, applies limits and records cost	LiteLLM, the AI gateway in Azure API Management, Cloudflare AI Gateway
Tools: coding agents	Agents that read, edit and run code in your repository	Claude Code, Codex CLI (command-line interface), GitHub Copilot, Cursor, Devin Desktop (formerly Windsurf)
Tools: agent frameworks	Libraries and SDKs (software development kits) for building your own agents	Claude Agent SDK, OpenAI's Agents SDK, Google's Agent Development Kit, Microsoft Agent Framework, LangGraph
Tools: the protocol	One standard way for AI applications to connect to tools and data	MCP

The layers cross. Anthropic's models page lists model IDs (identifiers) for Amazon Bedrock, Google Cloud and Microsoft Foundry as well as its own API, and Amazon Bedrock and Azure OpenAI in Microsoft Foundry both carry OpenAI's flagship model. Google and OpenAI appear in both rows of makers, with closed flagship models and open-weight ones. So the channel is its own decision. It settles where the keys live, whose bill the spending lands on, which contract and service-level agreement cover the calls, which security review applies, and which of the maker's features you get. The model can be the same whichever channel you pick; its features are not always. It's also where lock-in hides. A team that builds against one channel's own SDK and features has tied its code to that channel; a gateway of your own, the one door every call goes through, keeps that coupling in one place, so a change of model or channel is made and tested there rather than in every team's code. Providers still differ in how they call tools, return structured output and handle safety, so a switch still means running the tests again. Chapter 27 builds one. MCP sits under the tools: the same tool server can serve a coding agent and a product agent, whoever made the model behind them.

Read the table by its first two columns. The categories are what the rest of the book relies on. The third column is today's names, dated, and it's the part most likely to have changed by the time you read it.

What changes fast, and what doesn't

The names move. On 30 September 2026, Google's cloud channel was called Gemini Enterprise Agent Platform, with its product page noting that it was formerly Vertex AI; the editor once called Windsurf had become Devin Desktop; and Azure OpenAI sat inside Microsoft Foundry. Anthropic, OpenAI and Google each keep a deprecations page in their API documentation, with the dates on which older models stop answering. Models, prices and product names are the fastest-changing parts of this field, and every chapter dates them in its "Check before you rely on it" list.

What lasts is the work in the first two columns of that table, and the harness around whichever model you use. This book's thesis is that the harness is where an organization's investment should go: a gateway that outlives any one provider, checks that catch a bad change whoever wrote it, evaluations that decide whether a new model may replace the old one, and records that say what each team spent. Chapter 1 sets out the job family from a dated sample of postings. Part I gives you the vocabulary. Parts II to VII build the harness piece by piece, in a companion repository you can run. Chapter 30, "Zero to Prod: One Path from a Request to Production," assembles those pieces into one path from a request to production, and Appendix B, the kit, holds the files that path produces, ready to copy.

Check before you rely on it

- **The executive order**, as the White House fact sheet and the order's own text read on 30 September 2026. The order asks for proposed legislation within 60 days, so the statutory definition may change after that.

- **Every name in the table**, as its vendor's pages gave it on 30 September 2026, including two renames, Vertex AI to Gemini Enterprise Agent Platform and Windsurf to Devin Desktop, and Azure OpenAI listed inside Microsoft Foundry. Check a name on its vendor's page before you repeat it.
- **Which channel carries which model**, and which of the maker's features it supports, changes as channels add and retire them. On 14 October 2026, Anthropic's page for Claude in Amazon Bedrock listed structured outputs and the Files API among the features not supported there. Check the channel's own model and feature lists.
- **Licenses for open weights differ**. As of 30 September 2026, Mistral Large 3 and Muse Glimmer were available under the Apache License 2.0, and Llama 4 under Meta's own license. Read the license before you build on a model.
- **Job titles** are as postings used them in September 2026.

Keep this

- Architect and platform engineer often name one body of work. The employer picks the title; read the work.
- A model comes from a maker, reaches you through a channel, and is used through a tool. Each is a separate choice.
- Names and prices change. The categories, and the harness, last.
- The book says AI, as the field and the postings do.

What an AI Platform Engineer Does

- The five different jobs that share one job title, the same work under an architect's title, and how to tell which one a posting describes.
- What those jobs have in common, and which parts of them this book covers.
- A way to sample the job market yourself, with a script that does the counting.

The lab for this chapter: `git checkout ch01` in your copy of the companion repository (browse it on GitHub). Try it takes about 5 minutes. Its steps need no account and cost nothing. Reading away from a computer? The steps show the output that matters, so you can follow along without running it.

The problem: one title, several jobs

Here are three real postings for an AI platform engineer. All three were open in the United States on 22 September 2026.

- **Staff Software Engineer, AI Platform, at a large job site** (hiring for itself). The work is the foundational services behind the company's AI products. The requirements are about reliability and scale, with almost nothing specific to AI.
- **Senior AI Platform Engineer, at a digital health company.** The work is production machine learning: training models, serving them in real time, monitoring them, and running it all on Kubernetes.
- **Principal AI Platform Engineer, at an insurance software company.** The work is a platform for the company's own AI agents: a registry of the agents, a catalog of the tools they may use, and agents that act with the permissions of the person who started them. The posting expects daily use of two coding agents, Cursor and Claude Code.

Same title, three jobs. The first is backend platform work at a company that sells AI features; the sample counts it with infrastructure, the nearest of the kinds below. The second is machine learning operations. Only the third is about agents.

An **agent**, for now, is an AI model that calls tools in a loop until it reaches its goal or a stop condition. A coding agent, for example, reads a file, runs the tests, reads the result and decides what to do next. Chapter 3 takes one apart and runs it.

If you're moving into this work from platform, DevOps (development and operations), site reliability or backend engineering, the title is the first thing to get past. Before you study, prepare for an interview or write a posting of your own, work out which job is actually on the table.

This chapter is built from a sample of 45 postings, each read in full and coded: 33 on 22 September 2026, and 12 architect-titled postings on 30 September. A script produced the counts in its tables, and in the Try it section you'll run that script yourself. A few counts in the prose, such as how the six gateway postings divide and where the first 33 came from, come from reading the data file itself.

How it goes wrong

Most mistakes with these postings come from reading them like any other job ad. Five cost the most.

Matching on the title. You apply because the title says "AI platform," and the job turns out to be about serving models on GPUs (graphics processing units). You can usually see this in the posting before you apply: the requirements talk about training pipelines, inference (running a trained model to get its answers), GPU capacity and Kubernetes, and agents come up once or not at all. In the sample, agents appear in only 1 of the 5 infrastructure postings.

Studying the tool list instead of the work. Postings name products. In the sample, Claude Code is named in 9 postings, Cursor in 8 and GitHub Copilot in 6. Products change. The work around them carries over from one product to the next. One posting from a large IT (information technology) consulting firm is entirely about moving an enterprise search system from Microsoft Semantic Kernel to Microsoft Agent Framework. Its requirements are about what has to survive the move: the permissions, the quality of the search results and the citations in the answers.

Reading a count as market share. A count from a search tells you what the jobs you found ask for, not how common each kind of job is, because you chose what to search for. The sample behind this chapter searched on purpose for some kinds of job, so those kinds are over-represented in it. The symptom is a confident sentence such as "most AI platform jobs are enablement jobs," built on a search that went looking for enablement jobs.

Skipping the enterprise version of the job. Banks, insurers and manufacturers describe the same work in other words and on other stacks: a central AI team, a library of reusable pieces that other teams build on, and review gates an AI project must pass before it ships, often on Microsoft and AWS (Amazon Web Services) products, and sometimes under an architect's title. If you skip a posting because you don't recognize Copilot Studio or Bedrock, or because its title says architect, you may be skipping the job you want.

Counting one job twice. Staffing firms repost a client's description, and large employers post one role for several cities. The symptom is two or three listings with the same paragraphs. In the sample, a role posted for several cities counts once. Two postings may be staffing firms' reposts; both were kept, because a staffing firm is filling a real opening, unlike an aggregator whose employer can't be checked, and neither repeats another posting in the sample.

WATCH OUT

If you write postings, the same confusion runs the other way. A posting titled "AI platform engineer" that doesn't say which job it is can be read five different ways. Name the kind of job in the first paragraph.

How it works: five kinds of job

Each posting in the sample was coded against a **codebook**: definitions of five kinds of job and a list of yes-or-no signals, such as whether a posting asks for agents or for evaluation. Its first version was written before any posting was read, and it has grown once since, when the gateway signal below was added. Every posting got a primary kind of job, and a second kind where it clearly had one:

Kind of job	The core work	Postings (of 45)
Enablement	Shared AI capabilities, standards and governance for many internal teams	24
Coding agents	Coding agents and AI developer tooling in engineering workflows	7
Infrastructure	Running, serving or training AI at scale	5
Product agents	Agents or AI features inside a product that customers use	5
Applied	One AI system for one domain	4

Fifteen postings also carry a second kind. Here is what each kind looks like in a posting. What a working week looks like in each is in the introduction, under "A working week in each kind of job."

Enablement: the central team (24)

The largest group. An enablement team builds what the company's other teams use when they build with AI: access to models, shared tools, guardrails (limits on what AI may do), evaluation (measuring the quality of AI output), monitoring, and training for the people who use all of it.

- **A social media company**, a senior staff role on its generative AI platform: an LLM (large language model) gateway that model calls pass through, with usage limits and failover; retrieval (search that finds the documents a model needs) for teams across the company; and standards for evaluating, governing and observing generative AI systems.
- **A specialty insurer**: a library of reusable AI capabilities and patterns for teams across the company, design reviews with technical sign-off at what the posting calls "AI readiness gates," shared tooling for running evaluations, and a community of practice with office hours.
- **A mid-size manufacturer**: enablement run from the IT department, Microsoft first. It covers Copilot Studio agents and GitHub Copilot, a way to take in and triage ideas for using AI, and coaching for "citizen builders," people outside engineering who build their own tools.
- **A payroll and human resources software company**, an architect posting: the patterns and standards that builders across its service organization use for agent workflows, including when a person stays in the loop, with legal and compliance requirements built into the

patterns rather than into each workflow.

Guardrails appear in 22 of the 24 enablement postings, retrieval and AI observability (logs, traces and metrics for AI calls) in 19 each, and evaluation in 18. If you picture "AI platform engineer" as one job, this is the one the sample shows most often, with the search-term caveat from the previous section. It's also where the architects are: 11 of the 15 architect-titled postings are enablement.

Coding agents: the harness around the agent that writes code (7)

These jobs build the environment in which AI coding agents work on a company's own code: what the agent is told, what it's allowed to do, and the checks its work has to pass.

- **A customer data software company:** cloud development environments where agents can run the full test suite; connections from agents to CI, Slack, Datadog and GitHub through MCP (the Model Context Protocol, a standard way to connect AI applications to tools and data); the repository-wide agents.md instruction file (as the posting writes it; the convention is AGENTS.md) and a shared library of reusable instructions for agents, which the posting calls skills; and fixing the root causes wherever agents struggle.
- **An AI inference company:** an internal platform for AI-assisted development. It includes conventions for CLAUDE.md and AGENTS.md instruction files, with tooling that keeps them accurate; internal MCP servers with scoped access; hooks (scripts that run automatically at set points in an agent's work) and sandboxes (isolated environments that limit what an agent can reach and damage); a gateway with cost controls and audit logs; and measuring AI's effect on cycle time, review latency and change failure rate.
- **A health benefits company:** a system in which agents write, run and fix end-to-end tests, with feedback loops from local runs through CI into merge requests, so an agent can read a failure, fix it and resubmit.

All three describe parts of a **harness**: everything around the model that steers it before it acts and checks its work afterward. The idea is often put as "Agent = Model + Harness," the one-line summary of Vivek Trivedy's "The Anatomy of an Agent Harness" (LangChain, 10 March 2026), which Birgitta Böckeler quotes, with a link, in "Harness engineering for coding agent users" (martinfowler.com, April 2026). Chapter 4 takes a harness apart piece by piece.

Coding-agent work is also the second kind of job in 4 more postings, so it's primary or second in 11 of the 45. The coding-agent signal counts more, 17 of 45, because it marks any posting that asks for coding-agent work or fluency, whatever its kind: 8 of the 24 enablement postings do.

Infrastructure: keeping the models running (5)

These jobs run the models themselves: serving them, building training and evaluation pipelines, and managing GPU capacity. A large networking company's posting is typical: APIs that expose AI models to other teams, Kubernetes deployment across several clouds, monitoring and security compliance. Model serving or training appears in 4 of the 5 infrastructure postings.

Agents appear in 1, at a life-sciences data platform company whose platform also runs its customers' agents. None of the 15 architect-titled postings is this kind. This book doesn't cover this kind of job; more on that below.

Product agents: agents your customers use (5)

These jobs build the platform under agents that customers use inside a product.

- **A legal software company:** background agents inside its legal product, with review queues and approval gates, evaluations that must pass before a release is promoted, an MCP server, and cost accounting for each customer.
- **A design software company:** the platform behind its AI features, including evaluations for design generation, its MCP server, and the sandboxes and harnesses for the agents behind its prompt-to-app tool and its design agent.

Applied: one system, one domain (4)

One AI system for one purpose, run as something other people depend on. **A financial services company** wants a lead for a code security scanner powered by a large language model. **A data software company** wants agents and automation pipelines for its documentation and localization work. The third is **the IT consulting firm's** migration, described earlier. The fourth, an architect posting at **a federal healthcare IT consultancy**, sets the patterns, the evaluation plan and the safeguards across one agency's AI use cases.

These sit closest to a line the method below had to draw, because its screen leaves out roles that build AI applications. The line fell at who depends on the system. Each of these four runs one system that many teams use, or sets the patterns for several, while a role that builds one application for its own users, such as a regional bank's contact-center voice bots, was screened out.

What they share

Here is how often each signal in the codebook appeared across all 45 postings:

Requirement	Postings (of 45)
Agents	39
Guardrails: limits on what AI may do, and rules its output must meet	32
AI observability: logs, traces and metrics for AI calls	32
Evaluation: measuring the quality of AI output	30
Security or compliance	30
Retrieval: search that finds the documents a model needs	28
MCP	25
Cost control: token spend, spending limits, knowing who spent what	18
Coding agents	17

Requirement	Postings (of 45)
Model serving or training	17
Kubernetes	14
LLM gateway: one front door to the models, or routing between them	6 (a floor; see below)
Human approval of AI actions	5

Six things stand out.

Agents are the common thread, except in infrastructure. Agents appear in 39 of the 45 postings: 38 of the 40 postings outside infrastructure, and 1 of the 5 inside it.

After agents, the work is making AI measurable and safe. Guardrails, observability, evaluation and security each appear in two-thirds or more of the postings.

MCP is a platform concern. The protocol's documentation describes MCP as "an open-source standard for connecting AI applications to external systems." It appears in 25 postings, and several of them describe governing MCP across a company: scoped access, catalogs of approved tools, and agents that act with the permissions of the person who invoked them. Chapter 12 builds an MCP server, and chapter 13 covers governing them across an organization.

Human approval is rare by name, and architects ask for it. Only 5 postings ask for human review or approval of what agents do: the legal software company's product, and four architect postings, among them a bank's enterprise agent platform. The architect postings treat approval as a standard to set: when a person stays in the loop, how they override or escalate, and safeguards in proportion to risk. That count shows what postings list. What the work needs is a separate question, and chapter 19 makes the case for building approval in early.

Gateways are rare by name. The codebook's newest signal, added on 30 September and coded for all 45, is an **LLM gateway**: one front door to the models a company uses, which holds the keys, enforces limits and tracks cost, or routes each request to a model by cost and capability. Six postings describe one, four as a front door and two as routing, among them the social media company's and the AI inference company's. Read the six as a floor, because for 23 of the first 33 postings the text couldn't be read again, and the code came from notes written before the signal existed. Chapter 27 covers both kinds.

Languages. Python is named in 33 postings, TypeScript or JavaScript in 13, Go in 12, .NET in 6 and Java in 5. That's why this book's companion repository leads with Python and keeps a TypeScript package alongside it. Where the Java and .NET mentions came from is its own lesson, told in the field note below. Across all 45, none of the Java or .NET mentions comes from a company job board.

The same work, titled architect

Fifteen of the 45 postings carry "architect" in the title. Three turned up in searches that never used the word, and fell into the same kinds with the same signals, so the sample was extended with searches for architect titles. The introduction says what the two titles mean, under "One

body of work, two titles." Here is what the sample shows.

- **The same kinds of job, mostly enablement.** Eleven of the 15 are enablement, 2 applied, and 1 each coding agents and product agents. None is infrastructure.
- **The same signals, weighted differently.** Against the 30 other postings, architect postings ask more often for retrieval (all 15, against 13 of the 30), guardrails (13 of 15, against 19 of 30) and a person's approval of what agents do (4 of 15, against 1 of 30). They ask less often for cost control (2 of 15, against 16 of 30) and coding agents (3 of 15, against 14 of 30), and name fewer languages: Python in 8 of 15, against 25 of 30.
- **A level up.** All 15 state the years of experience they want, with a median of 10. Of the other 30, 24 state it, with a median of 5.
- **Work in public.** Five of the 13 that could be read again for it ask in so many words for a public body of work, such as a public repository, published writing or a governance framework. Chapter 32 describes them.

In my reading, that's the same work described a level up: more patterns, standards and governance, and fewer tools, languages and bills. The cost-control gap may partly be wording, because several architect postings mention cost only in passing, which the codebook doesn't count. And the architect searches, like the others, went looking for this work, so they can't say what share of architect jobs do it. Most architect titles they found were screened out, chiefly vendors' customer-facing solutions architects, whose main job is designing a vendor's product into its customers' systems, often as part of a sale, rather than running a platform inside their own company.

What this book covers, and what it leaves out

This book covers the work shared by the four kinds of job that are about agents: enablement, coding agents, product agents and applied, whichever title a posting gives it. It follows both kinds of agent the front matter names: coding agents in your codebase, and product agents inside your application.

It leaves out the infrastructure job: training, fine-tuning (further training a model on your own data) and serving models, and running GPU clusters. That work has its own books. Chip Huyen's *Designing Machine Learning Systems* (O'Reilly, 2022) covers the design of machine learning systems, from training data to retraining and monitoring. Her *AI Engineering* (O'Reilly, 2025) is about building applications on foundation models (large, general-purpose models that applications build on): not the infrastructure job, but a good companion to this book. For serving and GPU capacity, start with the documentation for the stack your target posting names.

Where to start, by the job you want:

If you want	Start with
Enablement	Part VII (running it across an organization), then Parts IV and V
Coding agents	Parts II, IV and VI (context, guardrails, feedback loops)
Product agents	Parts III and V (tools, evaluation), then chapter 19

If you want	Start with
Applied	Parts III and V
Infrastructure	The books above

Every path through this book starts with Part I, chapters 1 to 5, and chapter 32 turns each row into an order through the whole book.

The method, so you can repeat it

The sample was built in five steps. You can use the same steps for the jobs you're considering.

1. **Find.** Search in more than one place, and write down every search exactly as you typed it. Every source below can be searched in a web browser; a tool only makes it faster. This sample was found in two dated steps, and its data file lists the searches as typed.
 - **22 September 2026, four sources:** Indeed, through a job-search tool, for engineer titles such as "AI Platform Engineer" and "AI Enablement Engineer"; companies' own job boards on three applicant-tracking systems (Greenhouse, Lever and Ashby); Indeed again in a web browser, for the postings the tool's rate limit had blocked, plus two searches aimed at employers outside tech; and one posting read from a copy of its text.
 - **30 September 2026, architect titles:** "AI Architect" on Indeed, remote and in New York, San Francisco and Chicago. After those four searches the tool refused more with a rate-limit message, so "AI Solutions Architect," "Enterprise AI Architect," "Agentic AI Architect," "AI Platform Architect," "GenAI Architect" and "LLM Architect" ran only as web searches restricted to the three job boards, beside scans of companies' boards for any title with "architect" and an AI word. Five architect postings found on Indeed earlier that day were added too; their searches aren't in the data file, so those five can't be found again the same way.
2. **Screen.** Decide what counts before you read. This sample's rule: the core work is building or running shared AI or agent infrastructure, tooling, guardrails or platforms that other teams, products or agents use. A title alone never qualifies, in either direction, and architect titles were screened by the same rule as engineer titles. Excluded: sales, solutions and forward-deployed roles (engineers placed with customers), AI-trainer gigs, roles that build AI applications rather than platforms, general software roles with a little AI in them, management roles, AI companies' roles building their own agent products, roles outside the United States, reposts by aggregators whose employer can't be checked, second postings for a team already counted, and postings that had closed. Write down every exclusion and the reason.
3. **Read.** Read every posting in full, from its original text, and keep a dated copy of that text, because postings close. The job-board postings in this sample were read as raw text from each board's own API, never through a summary, and one was read from a copy. A week later, 23 of the first 33 texts were gone, so the signal added then was coded from notes for those 23.

4. **Code.** Give each posting a primary kind of job, an optional second kind, thirteen yes-or-no signals and five languages, using the codebook's definitions. When a posting is borderline, choose the conservative code and say why in the posting's note.
5. **Count.** Let a script do the counting, and make it refuse to run if any posting is missing a field. A typo that silently counts as "no" is an error you would never see in the totals.

The sample is United States postings only, apart from one that lists Toronto as its location and states a United States salary range. If you work elsewhere, sample your own market with the same five steps: titles, stacks and the mix of kinds can differ from country to country, and a count from someone else's market can point your study the wrong way.

Here is one coded posting from the sample, abridged. The signals left out are all false.

JSON

```
{
  "id": "P26",
  "title": "Staff AI Test Automation Engineer",
  "posted": "2026-09-11",
  "source": "Indeed (browser)",
  "type": "coding-agents",
  "alsoType": null,
  "agents": true,
  "codingAgents": true,
  "evaluation": false,
  "typescriptOrJavascript": true,
  "codingAgentsNamed": ["Claude Code", "Cursor", "GitHub Copilot"],
  "yearsRequired": 10,
  "note": "Builds the system in which AI agents (Claude Code) generate, run and iterate on Playwright tests, with feedback loops from local runs through CI into merge requests so agents can interpret failures, fix and resubmit; quality gates on every merge request. Tracks the share of AI-generated tests that are correct on the first pass (not coded as evaluation, conservatively). Says agentic-tool experience 'is not optional.'"
}
```

The note shows step 4 at work. Tracking how many AI-written tests are right on the first try is arguably evaluation. It's a borderline case, so the conservative code is false, and the note says so.

UNDER THE HOOD

The codebook's first version is written before any posting is read, so the postings can't quietly reshape the categories. When a signal is added later, as the gateway was, it's defined first and then coded for every posting, old ones included, and each posting records where that code came from. The codebook is stored in the data file itself, under codebook, so anyone checking the counts can check the definitions too.

The script is `postings/tally.mjs` in the companion repository. Before it counts anything, it checks every posting, and if any field is missing or holds the wrong kind of value, it stops and lists every problem it found. That includes the three fields that may hold null, `alsoType`, `yearsRequired` and `degreeRequired`: a misspelled key would otherwise count as "the posting doesn't say." The check is a guardrail in miniature: a rule enforced by code, with an error message that says what to fix. Parts IV and VI do the same thing for AI agents.

Try it

CHAPTER 1 · TRY IT

An incomplete posting stops the count

IT WORKS

The tally counts your own sample, starting from the template's one placeholder posting.

```
node postings/tally.mjs my-postings.json
Included: 1. Excluded: 1. Pulled: YYYY-MM-DD.
```



BREAK IT ON PURPOSE

Delete one field from the placeholder posting.

```
my-postings.json
- "mcp": false,
```



CAUGHT

The tally names the missing field and counts nothing, so no number rests on incomplete data.

```
node postings/tally.mjs my-postings.json
my-postings.json: 1 problem(s). Nothing was counted.
- P01: "mcp" is missing. Set it to true or false. Codebook: Mentions the Model
  Context Protocol.
```

Try it in three steps: it works, you break it on purpose, and the break is caught.

You need the companion repository, cloned and set up as the front matter's "Getting started" shows (chapter 5 explains each step), and Node.js 24 or newer. Run each command from the repository's root folder.

1. Step to this chapter.

```
git checkout ch01
```

This gives you the repository at the commit this chapter's Try it was checked against, so your output matches the book's. Git prints a note about a "detached HEAD," which is normal after checking out a tag; `git checkout main` takes you back to the latest version. The tally needs only Node.js, so there's nothing to install.

2. Count the book's sample.

```
node postings/tally.mjs postings/sample-2026-09-22.json
```

The output starts like this, and the numbers match this chapter's tables:

```
# Tally of postings/sample-2026-09-22.json

Included: 45. Excluded: 38. Pulled: 2026-09-22.

Read on: 2026-09-22 to 2026-09-30 (2026-09-22: 33, 2026-09-30: 12).
Exclusion records by screening date: 2026-09-22: 16, 2026-09-30: 22.

### Primary type

| Type | Count | Share |
|---|---|---|
| enablement | 24 | 53% |
| coding-agents | 7 | 16% |
| infrastructure | 5 | 11% |
| product-agents | 5 | 11% |
| applied | 4 | 9% |
```

It goes on to count the signals, the languages and the coding agents named, then breaks each signal down by kind of job, by source and by the date a posting was read, sets the 15 architect titles beside the other 30, and ends with the experience and degree requirements. Its Excluded counts exclusion records, not postings: two of the 38 records stand for groups of seven and nineteen postings, and a few others list several, so at least 62 postings were screened out.

3. Start your own sample from the template.

```
cp postings/my-postings.template.json my-postings.json
node postings/tally.mjs my-postings.json
```

The output starts:

```
# Tally of my-postings.json

Included: 1. Excluded: 1. Pulled: YYYY-MM-DD.
```

The template holds the codebook and one placeholder posting. cp works in PowerShell and in macOS and Linux terminals. In the Windows Command Prompt, use copy postings\my-postings.template.json my-postings.json instead.

4. Break it on purpose. Open my-postings.json, delete the line "mcp": false, from the placeholder posting, save, and run the tally again:

```
node postings/tally.mjs my-postings.json
```

The output is:

```
my-postings.json: 1 problem(s). Nothing was counted.
- P01: "mcp" is missing. Set it to true or false. Codebook: Mentions the Model Context Protocol.
```

The script exits with code 1 and prints no counts at all, because counts built on incomplete data would be wrong in a way you couldn't see. Put the line back.

5. Code five real postings. Find five postings for jobs you would actually apply for. Replace the placeholder with them, one entry each, following the codebook's definitions, and record anything you read and left out under `excluded`, with the reason, one record per posting. Run the tally again. Which kind of job comes out on top? Which signals appear in more than half of your postings? Those signals are your reading list for the rest of this book.

Check that it worked. The tally exits with code 0 and prints your counts. To confirm the script behaves as this chapter describes, run its tests:

```
node --test postings/tally.test.mjs
```

The summary at the end reports `pass 15` and `fail 0`.

Field note: where a sample comes from changes what it says

Record where every posting came from, and read your counts by source before you believe them, because the source can decide the answer. This sample's first 33 postings came from four sources. A job-search tool supplied 11 before its rate limit stopped it, and one was read from a copy of its text. Company job boards on Greenhouse, Lever and Ashby gave nine more, all from technology companies, and none of them named Java or .NET. Searches on Indeed, two of them aimed at employers outside tech, added banks, an insurer, manufacturers, a consultancy and an aerospace company. Among those 33, all 4 Java mentions and all 5 .NET mentions come from Indeed or from the one posting read from a copy of its text, and none come from the company job boards. A sample drawn only from those boards would have shown no Java or .NET at all. The source wasn't the only thing that changed, though: the later searches also aimed outside tech, so record the search beside the source, or you can't tell which one moved the count. The tally prints counts by source for exactly this reason.

How to tell it's working

This chapter's skill is reading postings well. You can check it three ways.

- **You can place a posting.** For any posting, engineer or architect, you can name its primary kind of job in a few minutes and point to the two or three lines that decided it.
- **A script did your counting.** Your own sample has grown from the five postings in the Try it to ten or more for the kind of job you want, coded against the codebook, with every exclusion recorded and every count produced by the tally.
- **Your gaps are specific.** For each signal that appears in more than half of your target postings, you can point to something you've built or run that shows it. The signals you can't point to are your study plan.

Check before you rely on it

- **The sample is a snapshot.** It holds 45 postings, 44 in the United States and one in Toronto that states a United States salary range: 33 read on 22 September 2026 and 12 on 30 September. Postings close. This book describes each employer rather than naming it.
- **The searches weren't neutral.** They looked for coding-agent, enablement, evaluation and MCP roles, and on 30 September for architect titles. The counts describe what these jobs ask for. They can't tell you how common each kind is.
- **Product names are as the postings wrote them.** Check a product's current name in its vendor's documentation before repeating it in an interview.
- **The MCP definition** is quoted from the protocol's documentation as it read on 22 September 2026.
- **The gateway signal was coded from notes for 23 of the first 33 postings**, whose texts had closed by the time it was added on 30 September, so its count is a floor.
- **The five kinds of job are this book's own categories.** Another codebook would draw the lines differently. Say so if you quote the counts.

Talking about it in an interview

The likely question: "What does an AI platform engineer do?" or its cousin, "Why this role?"

A strong answer: "The title covers several jobs, so I read the work. Your posting describes an enablement role: a platform other teams build on, with guardrails, evaluation and cost control at the center. Some companies call that an engineer and some an architect; the work is largely the same. It isn't the infrastructure job, since serving models and GPUs hardly come up. Your top two requirements are guardrails and evaluation. For the first, I built [a check that fails a change and says what to fix], and it caught [how many] in [which weeks]. For the second, I'd walk you through [an evaluation I ran], with its date."

The likely follow-up: "How is this different from the platform work you've done?" "Most of it carries over: pipelines, gates, least privilege and measuring what changed. What's new is that the thing inside the pipeline answers differently every run and can act on its own, so the exact tests need evaluations beside them, a person's approval in front of what can't be undone, and a budget on every call."

Keep this

- "AI platform engineer" names at least five different jobs, and "AI architect" often names the same work a level up. Read the work, not the title.
- Outside infrastructure, agents are the common thread, with observability, evaluation, guardrails and security close behind.
- Study the work a posting describes. Its product names are the part most likely to change.
- Let a script do the counting, record where every posting came from, and read your counts by source before you believe them.