

FREE SAMPLE

AI Agents for Humans

*A Practical Guide for Tech Leads
Who Decide, Build and Run Them*

*What an agent is, the agent-washing test, the two
dials, the fit test and how to read a vendor demo*

Pranav S.

FREE SAMPLE · OCTOBER 2026

Contents of the full book

Everything in AI Agents for Humans, section by section. The parts marked INCLUDED are complete in this sample, and those marked EXCERPT are partly included. Every chapter also ends with prompts to copy, common mistakes, a 10-minute exercise and key takeaways.

How to Use This Book

Who it's for · About this book · The agent spec you'll build · How to read it · Conventions · Companion files

PART I: UNDERSTAND AGENTS

- | | |
|-----------|---|
| CHAPTER 1 | What an Agent Is, and When You Don't Need One EXCERPT |
| | <i>A working definition · Where the word comes from · What agents get confused with · The agent-washing test · Two dials · Where agents fit, and where they struggle · The ideas this book is built on · Meet Fieldline · What this book won't do, and how it's organized</i> |
| CHAPTER 2 | Inside an Agent: Its Parts and the Loop |
| | <i>The nine parts · One run, turn by turn · The loop, as a sketch · Tokens and cost per run · Stop conditions · Why runs differ · State outside the window</i> |
-

PART II: WHAT AGENTS CAN DO

- | | |
|-----------|--|
| CHAPTER 3 | What Agents Can Do Today |
| | <i>Judge the conditions, not the capability · Capability by capability · Checkability is the common thread · What changes fast · What agents still shouldn't do alone · A one-day test on your own task</i> |
| CHAPTER 4 | When Agents Beat the Alternatives, and When They Don't EXCERPT |
| | <i>The honest version of "why agents" · How an agent compares with the alternatives · The fit test, Gate 1: does it need an agent? · The fit test, Gate 2: can an agent do it safely? · Is your company ready? · Climb the ladder on paper first · Dividing the work with people · A candidate that fails: Collections</i> |
| CHAPTER 5 | Use Cases by Function |
| | <i>How to read this chapter · Seven functions · An industry lens, including regulated work · Patterns across the use cases</i> |
-

CHAPTER 6

Coding Agents: What They Teach, and Running One in CI

Why coding agents work as well as they do · Design lessons worth copying · Three forms, at the time of writing · A team policy for coding agents · Upgrade bot: a coding agent in CI · Using coding agents to build your agents

PART III: WHAT GOES WRONG, AND HOW TO PROVE IT DOESN'T

CHAPTER 7

How Agents Fail: A Field Guide EXCERPT

Why failures come before controls · Model failures · System failures · Security failures · Organizational failures · Finding the failures most likely for your agent

CHAPTER 8

Prove It Works: Evals First

Why demos mislead · Build the first set · What an eval case looks like · What to grade · Three kinds of grader · Run every case more than once · Reading the results · Grow the set, and run it in CI · Keep evals portable · What public benchmarks can and can't tell you

PART IV: DECIDE AND DESIGN

CHAPTER 9

Build, Buy or Managed Service EXCERPT

Five ways to get an agent · Hosted models or models you run yourself · Five criteria · The layers of the agent stack · Questions for a vendor · How to read a demo or a vendor case study · A bake-off with your own eval set · Running your eval set on a product you don't control · Switching cost is real · Fieldline's decision

CHAPTER 10

Tools: What Your Agent Can Do

Tools are an interface for a model · Three kinds of tool · Fewer, bigger tools · Descriptions that work · Strict input schemas · Results the model can use · Read tools and write tools · MCP: sharing tools across agents · Code execution as a tool · Test tools on their own first

CHAPTER 11

Context and Instructions

What the model sees · Writing the system prompt · Start from the procedures you already have · Examples · Getting knowledge in · Keeping the window clean · Load instructions on demand · Memory across runs · Long-running work · Prompt caching · Version prompts like code

CHAPTER 12

Patterns: From Workflows to Several Agents

Start with the workflow · The workflow patterns · When to switch to an agent · When one agent isn't enough · Choosing a model for each step · An evolution path, and hybrids · A decision table

CHAPTER 13

Oversight: What the Agent May Do on Its Own

The oversight dial, per action · Where these ideas come from · What always needs approval · Automated approval as a middle ground · Escalation the agent can actually do · Make review fast and real · Trust in both directions · The experience of the people who use it · Asynchronous approval · Learning from reviewers · Moving an action up the dial

PART V: RUN IT SAFELY

CHAPTER 14 **Security and Data**

The threat model in plain terms · Prompt injection · The lethal trifecta · Least privilege and agent identity · Contain the damage · Guardrails and their limits · Third-party tools and MCP servers are dependencies · Data and privacy · Decisions about people · Red-team before launch · Reference frameworks

CHAPTER 15 **Running Agents in Production**

Trace everything · The numbers to watch · Budgets and limits · Cutting cost without hurting quality · Reliability · Rolling out · When an agent causes an incident · Model and provider changes are migrations

CHAPTER 16 **Debugging and Improving Agents**

Read the trace first · From symptom to cause · The fix ladder · A Scout example: the same log query, over and over · Using a model to help · When to stop tuning

PART VI: PUTTING IT TOGETHER

CHAPTER 17 **Building Triage End to End**

How to read this case study · Step 1: Spec section 1, the problem and the fit · Step 2: The eval set first · Step 3: The bake-off · Step 4: Version 1 as a workflow · Step 5: Version 2 adds an agent for the long tail · Step 6: Oversight · Step 7: Security and data review · Step 8: Shadow mode, then a pilot · Step 9: The first month · What made this work

CHAPTER 18 **Leading Agent Work**

Who owns what · It's software: treat it that way · Plan in stages, and set expectations upward · Measuring value honestly · Governance without bureaucracy · The people whose work changes · The agent Fieldline didn't build · When to stop · A 90-day plan for a first agent

CHAPTER 19 **What Comes Next**

Signals already visible · Open problems · No-regret moves · How to stay current

AFTERWORD **Keep Going**

The short version · Where to go next · About this edition · The four books · One last thing

APPENDICES

APPENDIX A

The Agent Review Kit EXCERPT

How to use this kit · The blank agent spec · The fit test · The readiness check · The first ten questions to ask a vendor · Fifteen more questions before you sign · The eval plan · The oversight table · The tool design checklist · The system prompt skeleton · The security and data review · The cost-per-task worksheet · The design-review checklist · The launch checklist · The agent register entry · The 90-day plan · The coding-agent policy · An example spec: Triage

APPENDIX B

40 Prompts for Agent Work EXCERPT

Deciding and the fit test · Writing the spec · Vendors and demos · Building eval sets and rubrics · Designing tools and instructions · Reviewing a design or a trace · Security, in concept · Launch and operations

APPENDIX C

The Landscape at the Time of Writing

How to read this appendix · Agent SDKs · Managed agent services · Coding agents · Protocols and formats · Feature status · Recently retired or retiring · Product-specific data terms

APPENDIX D

Starter Kit for Builders

What's in the kit · The model wrapper · Triage's tools, answering from fixtures · The loop · The workflow: version 1 · The graders · The harness and the report · The sample cases and fixtures · What a run prints · Making it yours

APPENDIX E

Use-Case Cards

How to use these cards · The template · Card 1: Upgrading a dependency · Card 2: Refactoring the invoicing module · Card 3: Billing and invoicing tickets · Card 4: Investigating an alert · Card 5: Ad hoc data questions · Card 6: A brief before a sales call · Card 7: Matching unclear payments · Card 8: Overdue-invoice outreach · Card 9: Questions about product behavior · Card 10: Looking up one field

APPENDIX F

Troubleshooting and Glossary

Troubleshooting · Glossary

APPENDIX G

Sources

Sources, A to Z · Keys by chapter

What an Agent Is, and When You Don't Need One

By the end of this chapter you'll be able to say what an agent is in one sentence, tell a real one from a relabeled script in a vendor pitch, and describe any agent with two separate dials. You'll also meet Fieldline, the company whose agents run through the book, and write the first lines of your own agent spec.

A working definition

An agent is a model that decides its own next step, in a loop. It calls a tool, sees the result and decides what to do next, until it finishes the task or hits a limit you set.

Four terms, defined once. The *model* is a large language model, the kind behind any chat assistant. A *tool* is anything the model can ask your software to do, such as searching the help center or looking up a customer.

The model doesn't run tools itself: it asks, your code runs the tool, and the result goes back to the model. Each pass through that cycle is a *turn*, and the whole job, from first input to final stop, is a *run*.

The definition is clearest next to the two designs it's most often confused with. Take one support ticket: a contractor says their invoices went out at last month's prices, though they'd changed their rates that morning.

- **A single model call.** Your code sends the ticket to the model with an instruction (“draft a reply”) and gets text back. The model sees only the ticket, so it can apologize fluently but can’t find out what happened.
- **A fixed workflow.** Your code runs a set sequence: classify the ticket (a model step), look up the account and fetch the help article on changing rates (code), then draft a reply (a model step). The model does some of the steps, but your code decides which steps run and in what order.
- **An agent.** The model gets the ticket and a set of tools, and chooses. It might look up the account, see that the rate change was saved while the invoice run was in progress, search past tickets, find similar reports linked to a known bug, and *escalate*: hand the ticket to a person, with a summary. Another ticket would get different steps, chosen by the model.

Anthropic’s post *Building effective agents* (December 2024) draws this line between workflows and agents. It describes workflows as models and tools run through code paths you define in advance, and agents as models that direct their own process and their own use of tools. OpenAI’s *A practical guide to building agents* draws a similar line: apps where the model doesn’t control how the work runs, such as simple chatbots and single-turn calls, aren’t agents. Hugging Face’s documentation for its smolagents library describes agents as programs whose workflow the model’s output controls.

The test they share: **who decides the next step?** If your code decides, you have a workflow. If the model decides, you have an agent.

A single call

one request,
one answer



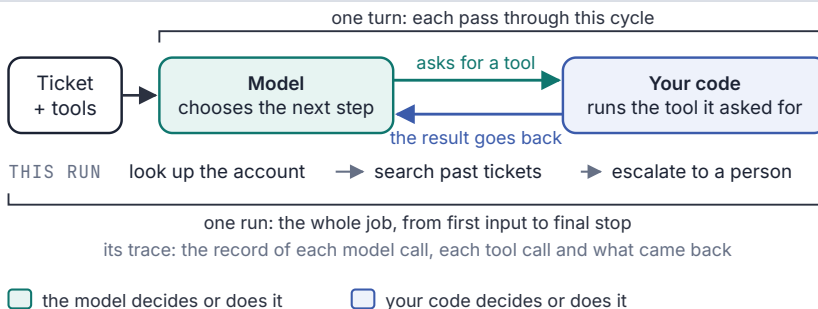
A fixed workflow

your code decides
the steps



An agent

the model decides
each next step



The opening ticket, handled three ways. In a fixed workflow your code decides the steps; in an agent the model decides each next step, and your code checks and runs each tool it asks for.

A workflow is cheaper, faster and easier to test, so don't reach for an agent by default. Anthropic's post makes the same point: start with the simplest solution that works, and add complexity only when you need it.

One question belongs with the definition from the start: **how will you check its work?** An agent that chooses its own steps can choose the wrong ones, and it reports back just as fluently either way. Chapter 8 turns that question into a practice.

The agent-washing test

Because "agent" sells, products get the label whether or not a model chooses anything. The analyst firm Gartner calls this *agent washing*: rebranding existing products, such as assistants, RPA and chatbots, that lack real agent capabilities. A second problem is worse: a real agent sold with no clear limits.

Five questions catch both. Put them to any vendor pitch, or to a colleague's demo, this week. Wherever you can, ask to see a *trace*: the step-by-step record of one run, showing each model call, each tool call and what came back.

Question	A real answer sounds like	A warning sign
1. Does the model choose the steps?	"The model decides which lookups to run, and in what order. Here's a trace."	"It follows the flow you set up." That's a workflow: fine, but price and test it as one.
2. What tools can it call?	A specific list, with what each tool can read or change, and with whose permissions	"It connects to everything."
3. What stops it?	Limits on turns, time and spend, and a handoff to a person	"It keeps going until the job is done."
4. Who approves what?	Settings per action: replies wait for approval, say, while tags don't	One switch for the whole agent, or "It's fully autonomous."
5. How is it evaluated?	"Run your own cases in the trial. Here's where it fails."	One polished demo, or an accuracy figure with no definition

Chapter 9 turns these into a full buyer's process, and Appendix A has 25 questions for a vendor: ten to ask before you agree to a trial, and fifteen more before you sign.

Two dials

Ask how autonomous an agent is and you're really asking two questions. This book keeps them apart as two dials.

The design dial: how much the model decides the steps. It's a ladder with four rungs:

1. **A single call.** One request, one answer.
2. **A fixed workflow.** Your code sets the steps, and the model does some of them.
3. **An agent.** The model chooses the steps, in a loop.

4. **Several agents.** Agents that hand work to each other, or to an agent that coordinates them.

Each rung up buys flexibility, and costs more model calls, more waiting, more ways to fail and harder testing. Climb only when the rung below can't do the job.

The oversight dial: how much a person checks. It's set per action, not per agent, using five *oversight modes*:

Mode	Meaning
Never	The agent can't take this action at all
Suggest	The agent proposes; a person acts
Approve first	The agent acts after a person approves
Act and report	The agent acts; a person reviews each action afterward
Act within limits	The agent acts inside set limits; a sample is audited

Chapter 13 shows how to choose a mode for each action.

The dials move independently. A fixed workflow, low on the design dial, can tag tickets on its own with a sample audited afterward: *act within limits*. An agent, higher up, can have its riskiest action set to *never*.

Upgrade bot, a coding agent at Fieldline (the company invented for this book, introduced later in this chapter), chooses its own steps to upgrade a dependency, but it can't merge anything: a person reviews and merges every pull request it opens. "Autonomous" squashes both dials into one word, so this book never uses "autonomy" without saying which dial it means.

OVERSIGHT DIAL
set per action

Single call

Fixed workflow

Triage

support tickets

Agent

Upgrade bot

dependency upgrades

Several agents

Scout

on-call alerts

Act within limits

a sample audited

Act and report

each reviewed after

Approve first

approved before

Suggest

a person acts

Never

no way to act

			 Post the incident summary
	 Tags, routing, escalation	 Open a pull request	
	 Replies		
	 Refunds, credits, account changes	 Merge; change CI; read secrets	 Restarts, rollbacks, config, paging

DESIGN DIAL how much the model decides the steps

Three of Fieldline's agents (Triage for support tickets, Upgrade bot for dependency upgrades, Scout for on-call alerts), with their oversight at launch. Each sits on one rung of the design dial, Triage at its starting rung, and its actions spread across the oversight modes, because oversight is set per action.

Published frameworks draw similar lines. Hugging Face grades agency by how much the model's output controls the program, close to the design dial, and AWS's security scoping matrix treats human oversight and the level of agency as separate dimensions. Chapter 13 credits these and others, and explains why this book uses named modes rather than another numbered scale.

The ideas this book is built on

Three jobs stay yours, however capable agents get: you decide what to build, you check that it's right, and you answer for it. The fit test and the agent spec are how you decide, evals are how you check, and because a person owns every agent, someone always answers for it.

Prove it with evals. The theme of the book fits in one sentence: an agent is done when it passes an eval set you trust, run more than once, not when the demo works. A demo shows one run on inputs someone chose; an eval set shows many runs on the inputs you actually get.

An *eval set* is a collection of real, anonymized cases (*eval cases*), each with the outcome a person agreed is right, plus *graders* that score each result: code, a model applying a rubric, or a person. Chapter 8 builds one, and Chapter 9 uses it to compare vendors.

The agent spec. The book's one artifact is a short document that describes one agent end to end. You'll write one for a candidate agent of your own, a section at a time, in most chapters' Try-it exercises.

Its eight sections cover the problem, users and fit; anatomy; likely failures and their controls; the eval plan and pass bar; build, buy or managed service; oversight; security and data; and operations. By the end you'll have a design you can take into a review or a budget meeting. The blank spec is in Appendix A.

The Four Rules. The spec enforces four rules, and you'll see them by name throughout:

- **Rule 1, Earn the autonomy.** Start low on both dials, and move up only on evidence.
- **Rule 2, Write the evals first.** Spec section 4 comes before any building or buying.
- **Rule 3, Design the context.** What the model sees decides what it does (spec section 2).
- **Rule 4, Bound the blast radius.** Limit how much damage any one mistake can do (spec sections 6 and 7).

Two more ideas sit under the Four Rules. **Code for the rules, the model for judgment:** if a step can be written as ordinary code, write it as code. And **a person owns every agent:** someone is accountable, by name, for its goal, its permissions and its failures.

NOTE: The full chapter also covers where the word comes from, what agents get confused with, where they fit and struggle, and Fieldline's five agents.

When Agents Beat the Alternatives, and When They Don't

By the end of this chapter you'll be able to run any candidate task through the fit test's two gates and say, with reasons, whether it needs an agent and whether an agent can do it safely. You'll also be able to compare an agent fairly with cheaper options, check your company's readiness, and treat a reasoned "no" as a result.

The fit test, Gate 1: does it need an agent?

The *fit test* is two gates that any candidate task goes through before anyone builds or buys. Gate 1 asks whether the work needs an agent, a model that decides its own next step in a loop, or whether code or a *fixed workflow* would do. In a fixed workflow, your code sets the steps and a model does some of them.

Answer each question yes, partly or no, from real cases rather than the demo:

1. **Do the steps vary from case to case in ways rules can't capture?** Signals: cases need different lookups in an order you can't predict, or a rules engine has grown too complex to maintain.
2. **Is the input unstructured, such as free text or documents?** Signals: emails, tickets, PDFs and notes that a person reads rather than a program parses.

3. **Is there a long tail of unusual cases?** Signals: odd cases keep arriving, and each needs something different.

If all three answers are no, use code or a fixed workflow. If a step can be written as ordinary code, write it as code: it's cheaper, faster and easier to test.

Any yes or partly sends the task on to Gate 2, but the three questions don't weigh the same. Varying steps is the strongest signal, because it's the one thing a fixed workflow can't handle. Unstructured input on its own often needs only a model step inside a workflow, and a long tail points to a hybrid: a workflow for the common cases, and an agent or a person for the tail.

So a task passes strongly when its steps vary across much of the work, and weakly when every yes sits in one corner, such as the replies to a routine message. A weak pass usually means a workflow, with a model step, an agent or a person for that corner.

OpenAI's *A practical guide to building agents* names three signals of its own: complex decisions, rules that have become hard to maintain, and heavy reliance on unstructured data. Without them, it says, a deterministic solution may be enough. Anthropic's *Building effective agents* recommends the simplest solution that works, and Google Cloud's guide to choosing a design pattern for agentic systems advises starting with a single agent rather than several.

Take Triage, the support-ticket agent at Fieldline, this book's hypothetical software company. Its steps vary only partly: most billing tickets need the same lookups, but a discrepancy can need three, in an order nobody can predict. Its input is free text, and odd cases keep arriving. That's a middle pass, neither strong nor weak: the varying steps sit in the long tail, so the design is a hybrid, with a workflow for the common tickets and an agent for the long tail.

The fit test, Gate 2: can an agent do it safely?

Gate 2 asks whether an agent can do the work without its mistakes reaching anyone. Answer each question yes, partly or no, and apply the last column's rule to every answer that isn't yes:

Gate 2 question	A yes looks like	If the answer is no
Can the result be checked?	A test, a rule, a second source or a person can tell right from wrong	Stop the project until you find a check
Is a mistake reversible?	You can undo it, or catch it before it reaches anyone	Every such action is approve first or never
Is the input trusted?	Only your own systems and staff write what the agent reads	Run the trifecta check (Chapter 14)
How many steps are there, and are there checks between them?	A few steps, or a check after each one that matters	Add checks, or split the task
Is there enough history to build evals?	Real past cases whose right outcome someone can label	Collect cases first (Chapter 8)

Approve first and *never* are two of the book's five oversight modes, set per action: the agent acts only after a person approves, or it can't take the action at all. The trifecta check looks for what Simon Willison calls the *lethal trifecta* (private data, untrusted content and a way to send data out, all in one agent) and removes at least one of the three. An *eval set* is a collection of real, anonymized past cases with agreed outcomes, used to test the agent before anyone relies on it.

A "partly" usually means the answer differs between actions or between kinds of case. Split them, and apply the rule to the part that answered no. A ticket's tags can be changed but a sent reply can't, so one agent can set tags on its own while every reply waits for approval.

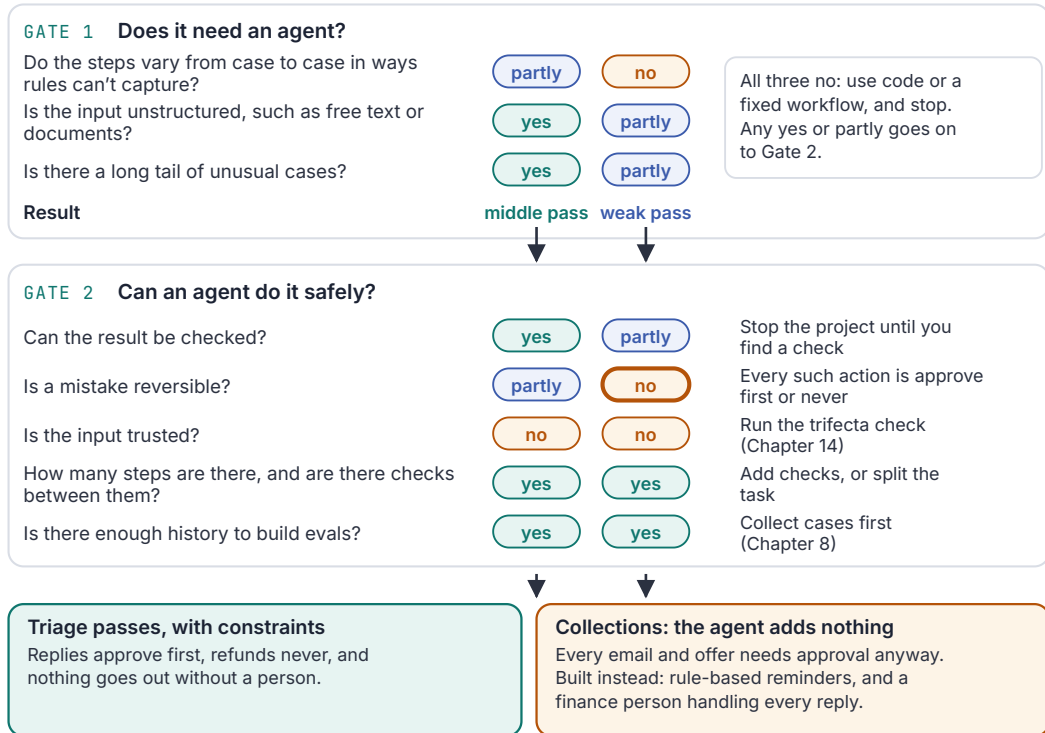
Gate 2 doesn't give a pass mark. A no on the first or last question holds the project until you have a check or the cases, and a no on the other three shapes the design. A candidate fails when those constraints leave the agent nothing

useful that a cheaper option can't do: when every action needs approval, say, and approving takes as long as doing the work.

For Triage:

- **Checkable: yes.** A support person reads every draft, and tags and routes can be checked against labeled tickets.
- **Reversible: partly.** Tags and routes can be changed, but not a sent reply or a refund. Replies are approve first; refunds and account changes, never.
- **Trusted input: no.** Anyone can write a ticket, and Triage reads account data, so it can't send anything out: a person approves every reply.
- **Steps and checks: yes.** A handful of steps, checks on every draft, and per-run limits on turns, tokens and time.
- **History: yes.** Enough resolved tickets for an anonymized eval set.

Triage passes, with those constraints built into its design. Appendix A has both gates and the scoring rule as a checklist.



Triage and Collections through both gates. Each answer that isn't yes adds its constraint; Collections' no on reversibility leaves the agent nothing to add, so Fieldline built a reminder workflow instead.

NOTE: The full chapter also compares agents with scripts, RPA, chatbots, workflows and people, checks whether your company is ready, climbs the ladder on paper, and runs Collections, a candidate that fails, through both gates.

How Agents Fail: A Field Guide

By the end of this chapter you'll be able to name the ways agents fail, at four layers, and trace each failure to the design choice that invites it. You'll also pick the five failures most likely for your own agent, with a control for each, which fills in spec section 3.

Model failures

These failures start with the model's choices inside a run: which tool to call, what to pass it, what to make of the result and when to stop. The examples come from Triage, the support-ticket agent at Fieldline, this book's hypothetical company.

Each row gives what you see, the usual cause, the control and the chapter that covers it. The cause ends with the choice that invites the failure, marked *Invited by*, because that's the part you can change. Two terms recur: the *context* is everything the model sees on a turn, and *evals* are anonymized past cases, each with an agreed right outcome, that you run the agent against (Chapter 8).

Failure and what you see	Usual cause	Control	Chapter
Calling the wrong tool. It searches the help center when the ticket needed account activity.	Vague or overlapping tool descriptions. <i>Invited by</i> one tool per API endpoint.	Fewer tools, each doing a whole job, described with when to use them; evals that grade the tools called.	10, 8

Failure and what you see	Usual cause	Control	Chapter
Bad arguments. A malformed account ID, a backward date range, an ID the model made up.	Loose schemas, or asking the model for values your code knows. <i>Invited by</i> free text where a fixed list belongs.	Strict schemas; code validates arguments before the tool runs; identity comes from code.	10
Ignoring a tool result. The activity shows a rate change saved during the invoice run, yet the draft says nothing changed.	The key fact is buried in a raw result. <i>Invited by</i> tools that return whole records.	Results with only the fields that matter; a grader that checks replies against tool results.	10, 8
Looping. The same search, again and again, until a limit stops it.	An empty result with no hint, so the model retries. <i>Invited by</i> no limit on repeated calls.	Results that say what to try next; per-run limits on turns and repeated calls.	10, 16
Stopping early. One lookup and a generic reply to a ticket that needed three.	Only the model decides when it's done. <i>Invited by</i> instructions with no definition of done.	Instructions that define a finished run; code that checks the output; evals that grade the steps.	11, 8
Claiming success without doing the work. It says it escalated, but <code>escalate_to_human</code> was never called.	Its own report is taken as the record. <i>Invited by</i> grading the final message, not the trace.	Code that checks what actually happened, in the trace and the systems touched.	8, 15
Confident wrong answers. A fluent reply citing a help article that doesn't cover the case.	The model fills gaps with plausible text. <i>Invited by</i> instructions to always answer, with no safe exit.	Claims rest on tool results; an escalation tool, and rules for using it; approve first on replies.	11, 13
Losing the thread in a long context. Late in a run, it forgets a constraint or repeats work.	Old tool results crowd out what matters. <i>Invited by</i> keeping everything in the context.	Trim old results, summarize history, and give side tasks to a separate agent that returns a summary.	11, 12

System failures

These failures start in the software around the model: the tools, the provider and the loop your code runs. They show up as stalls, duplicates, surprise bills and quality that shifts when nobody changed anything. The columns match the model table.

Failure and what you see	Usual cause	Control	Chapter
Context overflow. The provider rejects the request, or your code cuts older material, instructions included.	Big tool results and long histories fill the <i>context window</i> , the most the model can read at once. <i>Invited by</i> returning raw pages or whole records.	Truncate and paginate results; a token limit per run; summarize the history.	10, 11
Tool errors and timeouts. The run stalls, or carries on as if a failed call had worked.	Errors come back as stack traces, or as nothing. <i>Invited by</i> no timeouts.	A timeout on every call; errors that say what to try next; escalation after repeated failures.	10, 15
Runaway cost. One run, or one day, costs far more than usual.	The run so far is resent on every turn, so long or looping runs cost more each turn. <i>Invited by</i> no per-run token limit and no spend cap.	Per-run limits on turns, tokens and time; daily spend caps with alerts.	2, 15
Flaky dependencies. A case passes on Monday and fails on Tuesday, with nothing changed.	A downstream API or index that's slow, down or inconsistent. <i>Invited by</i> assuming dependencies are always there.	Tool results in the trace; a fallback to a simpler workflow or a person; evals run more than once.	15, 8
Retries that repeat an action. Two drafts on one ticket, or two escalations.	A call timed out after it had succeeded, and the retry did it again. <i>Invited by</i> write tools that aren't safe to repeat.	<i>Idempotent</i> write tools, which give the same result however often they're called.	10, 15
A model or provider change. Quality shifts with no change of your own.	The provider updated or retired your model version. <i>Invited by</i> unpinned versions and no eval set to rerun.	Pin versions; watch deprecation notices; treat each change as a migration and rerun the full eval set.	15

NOTE: The full chapter adds the security and organizational failures, and shows how to pick the five most likely for your agent.

Build, Buy or Managed Service

How to read a demo or a vendor case study

A vendor’s demo and its case studies each answer a narrower question than they seem to. A demo is one run, on inputs the vendor chose, with any failures left out. It shows the agent can succeed, and nothing about how often it does, so ask for what it leaves out (Chapter 1’s agent-washing questions apply too):

- **Run my cases.** Bring three of your own anonymized cases, including a hard one, and ask to see them run live.
- **Run it again.** The same input can produce a different run, so ask to see the same case twice more.
- **Show me a failure.** Ask what the agent gets wrong, and what it does then: stop, hand over to a person, or answer anyway.
- **Show me the trace.** A *trace* is the step-by-step record of one run: each model call, each tool call and what came back. Ask whether you’ll get the same view in production.
- **Show me the setup.** Ask what was configured for the demo, and how long that took.

A case study is a result the vendor reports, from a customer the vendor chose. It’s likely to show a good case rather than a typical one, so read it for what each figure measures:

The case study says	Ask
A “resolution rate” or “automated resolutions”	What counts as resolved: answered with no person involved, closed, or the customer simply didn’t reply?
“Productivity” or “time saved”	Per what, measured how, and by whom?

The case study says	Ask
An improvement	Over what baseline? With no baseline, there's no comparison.
Any figure at all	Was it before or after customization, and how much work did that take? Which kinds of case were in scope, and over what period?

Vendors' own guides to building agents can carry customer figures too. Anthropic's enterprise guide, *Building Effective AI Agents: Architecture Patterns and Implementation Frameworks*, cites results from several named customers, and this book repeats none of them. Treat such figures as a reason to test the agent on your own anonymized cases, several runs each: the bake-off later in this chapter.

APPENDIX A · EXCERPT

The Agent Review Kit

The first ten questions to ask a vendor

Send these in writing before any trial of an agent product, and ask for written answers. They cover what to know before any data goes in, even anonymized, and whether a trial can tell you anything. File the answers with your build-or-buy decision (spec section 5).

Data use, retention and training

1. What do you keep from our inputs and outputs, and for how long?
2. Do you train on our data, and can we opt out in writing?
3. Do this product's terms differ from your general terms?

Scopes and identity

4. Can each agent have its own narrowly scoped credentials, including read-only ones?
5. Does our system decide whose data the agent reads, or does the agent?

Traces and audit logs

6. Can we export full traces (every model call, tool call and result) and approval audit logs to our own store?

Approval controls and a kill switch

7. Can approval be set per action, rather than with one switch for the whole agent?
8. Can we stop the agent at once, and what happens to runs in progress?

Your eval set in a trial

9. Can we run our own anonymized cases, several runs each, in a sandbox, and see every result and trace?
10. Under what data terms do trial cases run, and are they deleted afterward?

NOTE: The full kit adds fifteen more vendor questions, the blank agent spec, the fit test, an eval plan, an oversight table, ten more templates and Triage's spec filled in as an example, all as editable files.

APPENDIX B · EXCERPT

40 Prompts for Agent Work

Forty prompts for any capable AI assistant or coding agent. Fill in anything in `<angle brackets>`, remove personal data and secrets before you paste, use an assistant your company has approved,

and have someone who knows the work check every answer.

Deciding and the fit test

Use these before anyone builds or buys. They lean on Chapter 4's *fit test*: Gate 1 asks whether a task needs an agent at all, and Gate 2 asks whether an agent can do it safely. A reasoned “no” is a result.

1. When you need candidates in one team.

PROMPT

From our <function> team's tasks below, pick the five best agent candidates, favoring read-heavy work, drafts a person reviews, and internal work. For each, give a good-fit version, its nearest poor-fit neighbor, and the question that separates them, such as reversibility.

<the tasks>

2. When someone asks “can an agent do this?”

PROMPT

Answer each question from my description only: yes, partly or no, with evidence, or "can't tell" and what you'd need.

Gate 1: Do the steps vary from case to case in ways rules can't capture? Is the input unstructured, such as free text or documents? Is there a long tail of unusual cases? If all three are no, recommend code or a fixed workflow and stop.

Gate 2: Can the result be checked? Is a mistake reversible? Is the input trusted? How many steps are there, and are there checks between them? Is there enough history to build evals?

Scoring: an uncheckable result stops the project; irreversible actions need a person's approval or are ruled out; untrusted input means no agent may also hold private data and a way to send data out; unchecked steps need checks or a split; no history means collecting cases first.

<the task, its actions, who checks it>

3. When you want the cheapest design that works.

PROMPT

Sketch this task as one model call, a fixed workflow where code sets the steps, and an agent choosing its own steps. For each, list the steps, mark the model steps and say what it can't handle. Sort my examples by the cheapest design that handles each. Which steps should a person keep?

<the task and examples>

4. When you're trying a task by hand first.

PROMPT

Can you do this task: <the task>? I'll paste anonymized examples one at a time, with what a lookup would return. Do the task, then list what you were missing or guessed, writing "unknown" rather than assuming. When I type "summary," group the failures by cause.

5. When the work is regulated or decides about people.

PROMPT

For the agent below, in <industry>, write questions for our legal or compliance team on the audit trail, explaining decisions, human sign-off, data rules, fairness, and telling people they're dealing with an AI. Phrase each as a question; don't say what's permitted or name laws.

<its inputs, outputs and actions>

NOTE: That's 5 of the 40 prompts. The full appendix adds prompts for writing the spec, vendors and demos, eval sets, tools and instructions, design and trace review, security and operations.

THE FULL BOOK

Get the Full Book

This sample is a small slice of the book. The full edition has 19 chapters and seven appendices, from deciding whether you need an agent to running one in production.

What you get

- The complete book as a PDF, with clickable bookmarks, that reads well on a laptop, tablet or phone
- All 19 chapters: the fit test, the failure field guide, evals first, build or buy, tools, context, patterns, oversight, security, production and a full case study
- Seven appendices: the agent review kit, 40 prompts, the product landscape at the time of writing, starter code, use-case cards, troubleshooting with a glossary, and the sources, with a key for each chapter that cites them
- Companion files: the review kit templates as editable Markdown, the use-case card template, and the starter code with sample eval cases
- A 30-day refund if the book isn't for you

How to buy

1. Go to aiagentsforhumans.com/ai-agents-for-humans and choose Get the book.
2. Pay through the secure checkout, run by Dodo Payments, which handles payment and any sales tax, VAT or GST.

3. Your download link arrives by email straight away, and you can download the book again any time from the Dodo Payments customer portal.

About the author

Pranav S. is an independent author. *AI Agents for Humans* is a practical, vendor-neutral guide for the tech leads, engineering managers and CTOs who decide whether to use an AI agent, buy or build one, and then run it.

Its claims about products, protocols, standards and published guidance were checked against primary sources as of October 2026.

AI Agents for Humans: Free Sample. A free sample of *AI Agents for Humans: A Practical Guide for Tech Leads Who Decide, Build and Run Them*, first edition. Checked against published sources as of October 2026. © 2026 Pranav S. You're welcome to share this sample with anyone, as long as it stays unchanged and free.

This is an independent, vendor-neutral book, not affiliated with, sponsored by or endorsed by any company it names. Products and published guidance change often; where anything here differs from the official documentation, the documentation is the source of truth.