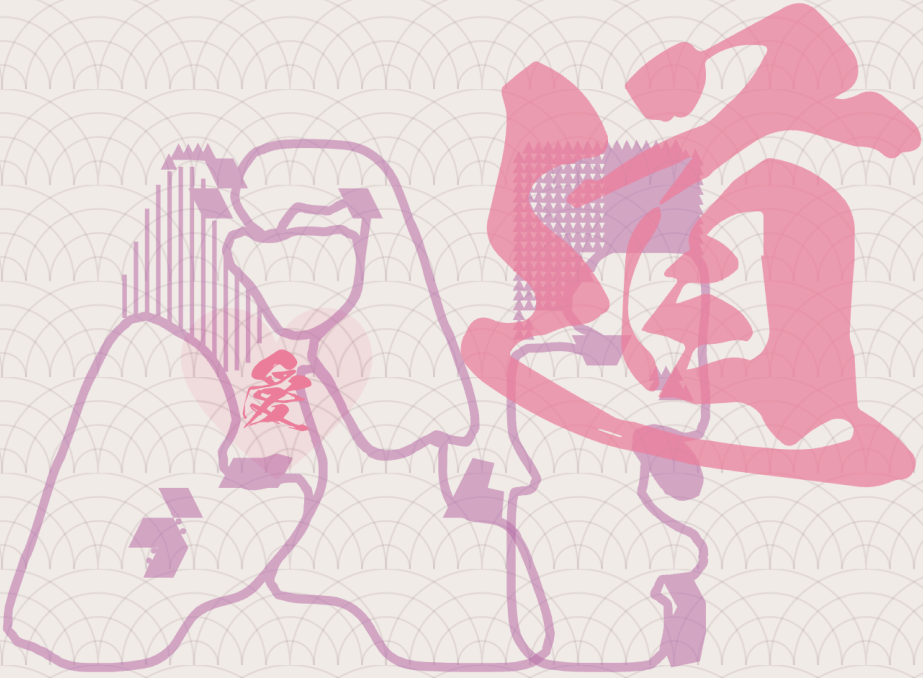
A STRUCTURED WAY OF WORKING WITH AI

AI-dō

The Way of AI, grounded in practice (道)



Chris Tham

helloTham

AI-dō — The Way of AI, grounded in practice (道)
Chris Tham

Read it online, or download the latest PDF and ePub:
christham.net/aidou
christham.net/aidou/ai-do.pdf · PDF
christham.net/aidou/ai-do.epub · ePub

This book is open source. The text, the code it discusses, and the full toolchain that builds the website, PDF, and ePub are on GitHub:
github.com/ChristineTham/aidou

Twelve beautiful illustrations by Katerina Limpitsouni
(unDraw · undraw.co), recoloured to the book's palette.

Edition 1.1 — released 24 July 2026.

First published 2026.

Contents

Figures

Diagrams

Tables

AI-dō

The Way of AI, grounded in practice

- 愛 **(ai)** — Japanese word for love: care for the people the work touches.
- 道 **(dō)** — *the way, path, or disciplined practice*: a craft refined over a lifetime, as in *judō* (柔道).

AI 道 (romanised *aidou*) represents *a disciplined way of working with AI, guided by human-centred principles*.

It rhymes with *Play-Doh*, and that is intentional: use AI like clay, shaping and reshaping one push at a time.

Preface — Why This Book Exists

Why this book

Working with artificial intelligence has become strangely easy to do and strangely hard to do well. Anyone can open a chat window and get an answer in seconds. Getting answers you can rely on — accurate, consistent, worth using — is much harder.

We all tend to use the same handful of frontier models, so it is not the model that separates good quality from slop. The *method* matters — the practised habits by which an experienced person turns a capable tool towards a dependable result, the way a practised hand and a beginner, given the same tools, turn out very different work. This book is about that method: a structured way of working with AI, where you start from what you actually want, improve the result over several rounds, and check it before you use it.

Figure 0.1: Using AI well is a practice you learn, not a tool you buy — which is what this book is for.

A quick word on the terms, since they move quickly. When I say *artificial intelligence*, I mean the present generation of large language models — systems trained on enormous quantities of text, and increasingly images and sound, that respond to plain-language requests with fluent prose, working code, and structured analysis (Zhao et al., *A survey of large language models*, 2023). The most capable are called *frontier models*: the handful of largest, most general systems from a few well-resourced labs, the ones that set the pace and that everyone else measures against (Anderljung et al., *Frontier AI regulation: Managing emerging risks to public safety*, 2023). ChatGPT, Claude, and Gemini are the familiar names; behind them sits a *foundation model*, a single large network trained once at great expense and then adapted to countless tasks (Bommasani et al., *On the opportunities and risks of foundation models*, 2021).

> **i** Note
>
> A few terms used throughout, defined plainly:
>
> - **Large language model (LLM)** — a network trained to predict the next word, which in scale yields fluent prose, code, and analysis.
> - **Foundation model** — one large model trained once, then adapted to many tasks.
> - **Frontier model** — the largest, most general foundation models that set the pace (ChatGPT, Claude, Gemini).

Why I wrote it

A few words about me. My name is Chris Tham. I run a boutique strategy consulting company, **Hello Tham**, and I also teach technology and information systems to undergraduate and master's students at **Torrens University Australia**.

This book grew out of my recent experiences using AI in my work, and documents what I have learnt. I had just finished delivering an AI strategy for a global client, and I was also teaching AI to my students. Consulting gives me the experience, and teaching makes me explain it. This book is the written version of that explanation.

Why now, and won't it date?

Why write this in 2026? Because method now matters more than the tools. Most organisations have AI somewhere in the building, yet few can point to real results rather than just activity (Stanford HAI, *The AI index 2026 annual report*, 2026; McKinsey & Company, *The state of AI*, 2025). And as the models improved, which one you use stopped being what sets results apart. What separates good results now is the work you do around the model: how you set up the task, what context you give it, and how you check what comes back.

That is also why the book should outlast its examples. The field moves in weeks, and a book of prompts and tool tips would be stale a model release later. This one rests on a steadier layer — *how* the models work, not which model leads this quarter — and those properties have held across generations of models. Where a point is tied to a 2026 product or figure, I mark it as such; the way of working around it is meant to last.

Who it is for

This book is for the thoughtful professional — a leader, a consultant, an analyst, a designer, a builder — who wants structured, effective use of AI rather than a bag of prompts. I assume

- you are numerate and can read a diagram, a code snippet, or a research paper when it helps;
- you have already used these tools, and seen both what they can do and how often they get things wrong;
- and you want to *use* AI well, not to build models.

None of this is essential, but a little more helps: some feel for how software and information systems are built and used, and enough Python or JavaScript to be able to review code generated by AI. You don't have to work in IT to get a lot from it.

Using AI well is a discipline, and it is the one this book teaches. My promise is practical: an approach you can start using tomorrow, and that keeps improving with practice.

How to read it

The book is six chapters, and they build. The first lays the foundations — what AI is, how it works, and what it cannot do. The middle chapters climb from personal productivity, to building software, to the disciplines that keep that work sound, then to responsibility and governance. The last turns to mastery — what stays human when the tools are this good.

Table 0.1: The six chapters, and the question each one answers.

Chapter	Theme	The question it answers
1	Foundations	What is AI, what is it not, where do we stand?
2	Productivity	How does it change individual knowledge work?
3	Software	How does it change building software?
4	Disciplines	What keeps that work sound at scale?
5	Responsibility	How do we govern it safely and fairly?
6	Mastery	What stays distinctly human?

Read it once in order, to see how the ideas rest on one another, then come back for the parts you need. Every claim is cited inline to a

primary source, so you can follow the trail yourself; where the evidence is thin, I say so.

How this book was made

This book is built the way it argues you should work, so it is worth a short note on how. Every chapter is a single plain-text file written in GitHub-Flavoured Markdown — the same lightweight format the models read and write best — with the diagrams drawn in Mermaid, a text notation for flowcharts and timelines. There is one source. The website you may be reading, the PDF, and the ePub are all generated from those files; nothing is laid out twice by hand.

The look comes from a single file of *design tokens*. Every colour, type size, and per-chapter accent lives in one `theme.yml`, and a small script turns it into the separate stylesheets each format needs — so adjusting a heading weight or a shade of pink is one edit that flows to the website, the PDF, and the ePub at once. The palette is Rosely, a set of soft, Pantone-named tones: a near-black ink it calls *black-beauty*, *radiant-orchid* for accents, *raspberry-sorbet* for links, chosen to stay calm across a long read. The type pairs Raleway for headings — its heaviest Black weight for titles — with Noto Serif for the body and Noto's Japanese cut for the two kanji, 愛 and 道, that give the book its name. Both families are open-licensed and travel with the book, so it renders the same on any machine.

The build is Quarto. It renders the Markdown to a searchable website and, through the Typst typesetting engine, to a six-by-nine-inch PDF laid out like a printed trade book, and to a reflowable ePub for e-readers. The Mermaid diagrams are tinted to the same Rosely palette in every format, so a flowchart looks the same on the web, on the page, and on an e-reader. Push a change to the repository and a GitHub Action runs the same pipeline and publishes the result to the web.

I designed the cover myself, using AI to help create it in SVG — a text format a model can write directly. The design has three parts: the title *AI-dō* set in Raleway Black, a crumbling, glitching "AI" in the display face *Rubik Glitch Pop*, and, sweeping over both, the character 道 in *KokuryuSou*, a brush font of raw, dragon-like strokes. A heart-shaped 愛 is hidden inside the "A". According to Chinese astrology, I am a wood dragon. The lettering is then flattened to vector outlines, so the finished file needs none of those fonts to display; a headless browser turns it into the cover image, and the same artwork, re-laid out wide, becomes the card you see when the book is shared online.

The Japanese motif is not confined to the cover; it runs through the whole book, and it is personal — the fruit of three years spent learning

the language and immersing myself in the culture. The title is the seed: *AI-dō*, where 道 (*dō*) is the *way*, the lifelong disciplined practice behind arts like *judō*, and 愛 (*ai*) is love, the care for the people the work touches. Both characters, brushed in that same dragon-stroke hand, open every chapter over a *seigaiha* wave pattern drawn from classical Japanese design. The theme reaches into the argument as well. Chapter 2 is subtitled 愛 *in practice*, and Chapter 6 closes on *shuhari* (守破離) — the martial-arts progression from keeping the form, to breaking it, to leaving it behind.

Much of this toolchain was built the way the book describes: I said what I wanted, let the tools work out how, and checked what came back. The whole toolchain is open source, alongside the text, at github.com/ChristineTham/aidou.

References

Anderljung, M., et al. (2023). *Frontier AI regulation: Managing emerging risks to public safety*. arXiv. https://arxiv.org/abs/2307.03718

Bommasani, R., et al. (2021). *On the opportunities and risks of foundation models*. arXiv. https://arxiv.org/abs/2108.07258

McKinsey & Company. (2025). *The state of AI*. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

Stanford Institute for Human-Centered AI. (2026). *The AI index 2026 annual report*. Stanford University. https://hai.stanford.edu/ai-index/2026-ai-index-report

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., et al. (2023). *A survey of large language models*. arXiv. https://arxiv.org/abs/2303.18223

1 Foundations (The Way)

This chapter lays the foundations: what AI is and how it got here, where the field stands today, how a language model actually works, how the tools climbed from chatbot to agent ecosystem, what they still cannot do, how I came to trust them, the mental model I recommend, and the principles that follow. The rest of the book builds on these principles.

1.1 What AI Is, and How It Got Here

Artificial intelligence is older than the chatbots that made it famous. The term was coined in a 1955 proposal for a summer workshop at Dartmouth College, whose authors held that "every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it" (McCarthy et al., *A proposal for the Dartmouth summer research project on artificial intelligence*, 1955). The decades that followed ran hot and cold. Early systems encoded human knowledge as hand-written rules — *expert systems* that worked inside a narrow domain and broke outside it — and twice the field's promises outran its results badly enough to trigger an "AI winter" of lost funding and faith (Nilsson, *The quest for artificial intelligence: A history of ideas and achievements*, 2010).

A different idea ran quietly alongside the rules: let a machine learn from data instead of being told the rules. Frank Rosenblatt's *perceptron*, a simple mathematical model of a neuron, dates to 1958 (Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain*, 1958), but neural networks stayed a backwater for decades, starved of the data and computing power they needed. That changed in 2012, when a deep network — many layers deep, each learning features at a higher level of abstraction — won the ImageNet image-recognition contest by a wide margin, moving *deep learning* to the centre of the field (Krizhevsky et al., *ImageNet classification with deep convolutional neural networks*, 2012).

The architecture that made language work arrived in 2017, when researchers at Google introduced the *transformer*. Its *attention* mechanism lets the model weigh every word against every other, so it could learn the long-range structure of text far better than anything before

it (Vaswani et al., *Attention is all you need*, 2017). The rest came from scale. Train a transformer to predict the next word across enough text and it becomes a *large language model*, and once it is large enough, it turns fluent. That is the path that leads to ChatGPT in late 2022, and to everything in this book. The preface set out the terms — *large language model*, *frontier model*, *foundation model* — and this chapter explains what is going on underneath them.

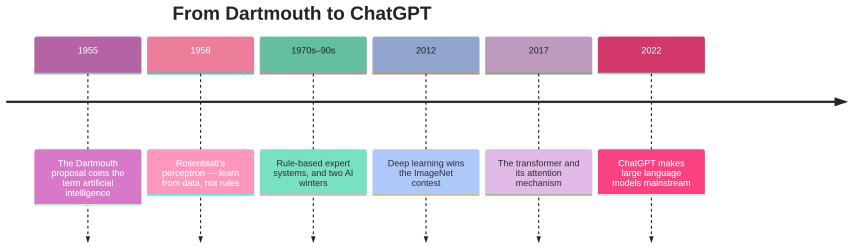


Diagram 1.1: Seventy years from the Dartmouth proposal to ChatGPT.

I actually used a language model in 2018 whilst consulting for a government client. Lesson 10 in fast.ai's Practical Deep Learning for Coders course described ULMFiT, a groundbreaking approach to *transfer learning* — reusing what a model learned on one task as the starting point for another — in natural-language processing. The recipe: take a language model pre-trained on Wikipedia to predict the next word, strip away its final layer, and repurpose it into a highly accurate text classifier. I adapted the lesson's codebase to solve a real-world enterprise problem: automated support-call classification. I built the pipeline on AWS SageMaker, *fine-tuned* the base model — trained it further — on the client's call logs, and it sorted support calls into categories (Procurement, HR, IT, and so on). Even then, the part that made it useful was already the part this book is about: adapting a general-purpose model to a specific problem, rather than the model itself.

1.2 The AI Landscape in 2026

Two facts about 2026 matter for everything that follows. The first is that these tools are everywhere: roughly 88% of organisations report using AI, even as most are still experimenting rather than depending on it (Stanford HAI, *The AI index 2026 annual report*, 2026; McKinsey & Company, *The state of AI*, 2025). The second is that the leading labs no

longer treat the model alone as the product; they now compete on the system around it: workflows, memory, and the cost of running it well.

Table 1.1: Three signals of where AI stood in 2026.

Signal (2026)	Figure	What it means
Organisations using AI	~88%	Access is near-universal
Coding benchmark cleared in a year	~60% → nearly all	Capability is accelerating
Organisations reporting real value	a minority	Value, not access, is the scarce skill

Sources: Stanford HAI, 2026; Benaich, *State of AI report 2025*, 2025.

Models can do far more than they could a year ago, but what they can do is patchier than ever. Getting an answer is now easy. Deciding whether the answer is any good is harder.

1.2.1 The open-weight frontier

For most of this story the frontier was closed and American: you rented the best models from OpenAI, Anthropic, or Google, and you could not see inside them. That is no longer the whole picture. Several of the strongest 2026 models are *open-weight* — their trained weights are published for anyone to download and run. Alibaba's Qwen 3.6 ships under a permissive Apache-2.0 licence (Qwen Team, *Qwen3.6*, 2026); Moonshot's Kimi K3, at 2.8 trillion parameters, is the largest open model released so far and, on its makers' own benchmarks, trades blows with the leading closed systems (VentureBeat, *China's Moonshot AI releases Kimi K3, the largest open-source model ever*, 2026). Two things stand out. The open models trail the closed American frontier by months, not years — a gap of roughly three to six months that has held steady rather than widened, with the top open model on one independent index sitting about five points below the best closed one (OpenRouter, *The open weight models that matter: June 2026*, 2026). And that open tier is now led by Chinese labs rather than an American one: Chinese models already make up something like two in five downloads on Hugging Face, ahead of American ones (Atalan, *What to know about Chinese AI models*, 2026).

That changes what a careful professional can do. When the weights are yours to download, you can run a frontier-class model on your own hardware or in your own cloud. That buys three things a rented API

cannot: your data never leaves your control, the cost is the machine rather than a per-token meter, and no single vendor's price rise, outage, or withdrawal can stop your work. The last of these is not hypothetical — §5.2.3 tells how a US export-control order switched Anthropic's Fable 5 off for every foreign user overnight. This book already suggested keeping an open-weights model in reserve (§5.6); the change is that the reserve can now be frontier-class.

None of this is free of risk, and the risks pull in different directions. The most immediate is political: the United States is moving to keep Chinese models out of government hands, which §5.2.3 takes up. The subtler risk travels inside the weights. Because these models are trained under Chinese content rules, their political alignment ships with them. A benchmark of sovereignty questions found every Chinese model it tested failing, often answering *worse* in English than in Chinese — tuning aimed at foreign audiences, the author suggests. The slant even survived running the model locally rather than through an API (Ko, *Bilingual bias in large language models: A Taiwan sovereignty benchmark study*, 2026). Read that as a direction rather than a verdict: it is a small study, and one with its own standpoint. Two cautions round it out — cheaper, leaner training tends to buy thinner safety testing, and leaning on any one country's models, American or Chinese, is its own kind of dependence (Atalan, 2026).

> **i** Note
>
> **The open-weight frontier, in brief.**
>
> - **Strengths** — frontier-class capability you can download and run; permissive licences; low, predictable inference cost.
> - **Weaknesses** — political alignment and censorship baked into the weights; thinner safety testing; specifications often self-reported.
> - **Opportunities** — data sovereignty, self-hosting, and independence from any one vendor or government.
> - **Threats** — moves to ban Chinese models (§5.2.3); trust and provenance concerns; trading a vendor's lock-in for a country's.

1.3 How a Language Model Works

Underneath the surface, a large language model does one thing: it predicts the next token. Given all the text so far, it guesses the next word, then the next, then the next. It was trained on vast amounts of text, so it has learned the probability of each possible next token, and it writes by sampling from those probabilities one token at a time. Stephen Wolfram

puts it plainly: the system is always just "adding one word at a time," picking a reasonable next token and, with a little randomness in the sampling, favouring variety over the single likeliest word (Wolfram, *What is ChatGPT doing … and why does it work?*, 2023). Nothing in the mechanism consults a fact store or checks whether the result is true. It simply produces the most plausible continuation.

> **i** Note
>
> A **token** is the unit a model actually reads and writes — not quite a word, but a common chunk of text: a whole word like " the", a word-piece like "pre" or "ing", a punctuation mark, or a lone character. Before the model sees anything, text is split into tokens drawn from a fixed vocabulary of tens of thousands, which is why a model can coin new words, and why an unusual name or a long number can cost several tokens each. A rough rule of thumb for English: one token runs about four characters, or three-quarters of a word — so token counts, which is what context limits and bills are measured in, never quite match word counts.

This raises an obvious question: if the model only guesses the next token, how does it reason at all, and why does it so often seem intelligent? The start of an answer is that predicting the next token well is harder than it sounds. To guess the next move in a game, the next line of a proof, or the next clause of a contract, the cheapest strategy available to a large enough network is to build an internal model of whatever produced the text, rather than memorising surface patterns. The cleanest demonstration comes from an experiment that trained a small GPT-style language model to do one thing: predict legal moves in the board game Othello. The model was given no rules and no picture of the board. Yet it developed an internal representation of the board state — one researchers can read out, and even edit to change its moves, which proves the model actually uses it (K. Li et al., *Emergent world representations: Exploring a sequence model trained on a synthetic task*, 2023). Wolfram frames the same surprise at the level of language itself: a next-token predictor can write a passable essay because doing so turns out to be "computationally shallower" than we assumed — human language is more regular and law-like than it looks, and the model implicitly discovers those regularities in training (Wolfram, 2023).

That hidden structure is what reasoning draws on, but a single pass through the network is a shallow computation: the model must answer the moment it stops reading. Letting it write intermediate steps first — a *chain of thought* — changes what the model can do, because every token it emits becomes input it can condition on next, so a hard problem

can be spread across many small, reliable steps instead of one leap (Wei et al., *Chain-of-thought prompting elicits reasoning in large language models*, 2022a). This is a real change in what the model can compute. A transformer forced to answer immediately provably cannot solve some strikingly simple problems — whether two nodes in a graph connect, or what a small state machine does — that the very same transformer *can* solve once allowed a scratchpad, because the intermediate tokens genuinely extend its computational reach (Merrill & Sabharwal, *The expressive power of transformers with chain of thought*, 2024). And the reason it works traces straight back to prediction: human writing comes in overlapping local clusters, so a model trained to predict it learns reliable short hops between related ideas and chains them into conclusions it could never reach in one step (Prystawski et al., *Why think step by step? Reasoning emerges from the locality of experience*, 2023).

Stack enough of this and whole abilities appear to switch on with scale — multi-step arithmetic, transliteration, chained logic that smaller models simply lack (Wei et al., *Emergent abilities of large language models*, 2022b). It is tempting to read that as understanding finally arriving. The cautious reading, and the better-supported one, is that much of the surprise comes from how we keep score: grade a task all-or-nothing and a steadily improving skill looks like a sudden leap, but measure it on a smooth scale and the cliff often flattens into a slope (Schaeffer et al., *Are emergent abilities of large language models a mirage?*, 2023). So here is the simple answer, which the rest of this chapter sharpens: the model reasons by building and chaining the structure that prediction forced it to learn. That reasoning is real and useful. But it is not the same as knowing — the model has competence without comprehension — and a wrong answer reads just as smoothly as a right one.

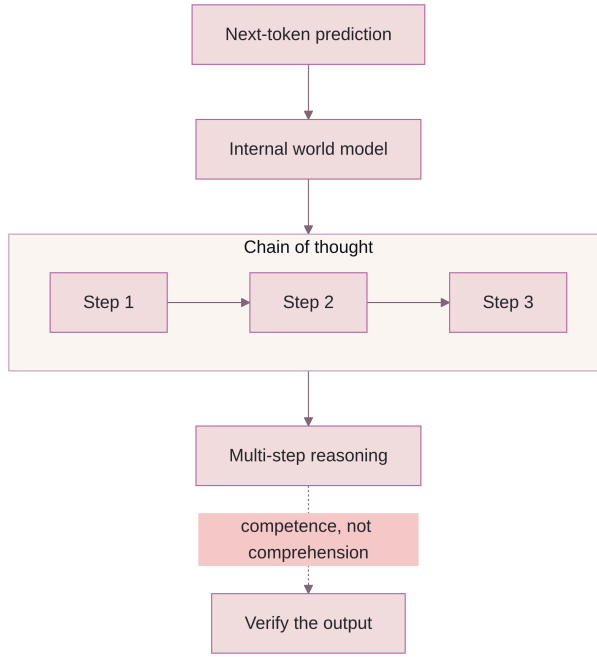


Diagram 1.2: How next-token prediction gives rise to real reasoning.

1.4 From Chatbot to Agent Ecosystem

The tools changed fast, and at each step the kind of work you could trust to them changed too. The story runs from a closed chat box to an ecosystem of agents that can take actions on your behalf.

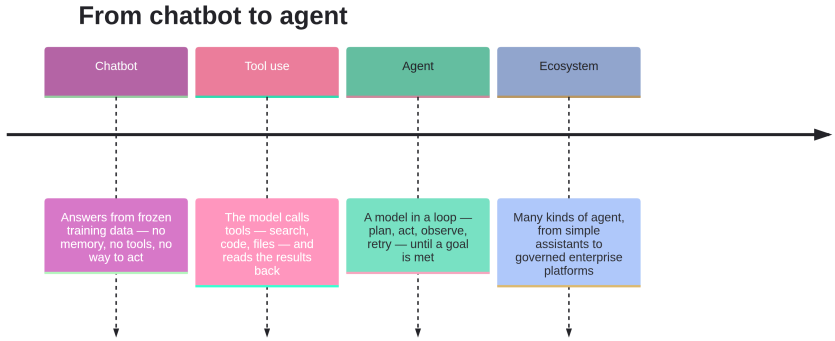


Diagram 1.3: The climb from chatbot to tool use to agent to ecosystem.

The first mass-market chatbots were articulate but frozen: they answered only from what they had absorbed in training, with no memory of you, no live information, and no way to act in the world. The first real change was giving the model hands — a way to recognise that a task needs a tool and to emit a structured call for it, so it could search, run code, or query data instead of guessing. The predictor of text became something that could also take actions and read back the result.

At first each connection was hand-built, one wiring job per tool; that soon gave way to shared standards, so a model could plug into tools and data through common interfaces rather than custom code. Those standards have since multiplied into an interoperability stack — one survey catalogues several and lays them out as stages to adopt, from tool access, to structured messaging, to one agent delegating to another, to open agent discovery (Ehtesham et al., *A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP*, 2025).

Tools are what turn a predictor of text into an *agent*: a model placed in a loop and allowed to plan, call a tool, read what comes back, and try again until a goal is met. The research community named the shift early, recasting the language model as the *brain* of an agent — wrapped in perception and action — and charting the climb from single agents to cooperating multi-agent systems (Xi et al., *The rise and potential of large language model based agents: A survey*, 2023).

> **Note**
>
> An **agent** is an LLM running tools in a loop to reach a goal (Willison, *I think "agent" may finally have a widely enough agreed upon definition to be useful jargon now*, 2025). A **tool** is an action it may take — web search, code execution, file edits — and the **loop** runs until a stopping condition is met. An agent has no agency in the moral sense: a computer cannot be held accountable, so you stay responsible for what it ships.

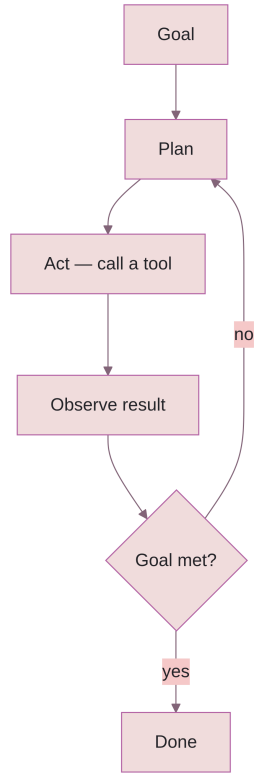


Diagram 1.4: The agent loop: plan, act, observe, and retry until the goal is met.

Today, that single loop has fanned out into a whole ecosystem. The same pattern scales from an assistant anyone can open in a browser to a governed platform running fleets of agents across an organisation;

what changes along the way is how much you hand off, and how much machinery it takes to do so safely.

> **i** Note
>
> **The tiers of AI tool, from least to most autonomous.** Each step hands off more, and takes more machinery to do it safely.
>
> - An **assistant** answers from its training and the context you supply. It has no autonomy of its own — you drive, asking and judging each reply.
> - A **tool-using assistant** calls tools on request — search, code, files — one step at a time, with you approving each action.
> - An **agent** loops towards a goal and checks its own work, running a whole task while you set the goal and review the result.
> - A **personal agent** is an always-on agent for one person, on your own machine or account, writing its own skills and building a model of your work. It runs continuously for you; you delegate, then check what it ships.
> - A **multi-agent system** has several agents coordinating, one delegating to another, running many tasks at once while you supervise the fleet.
> - An **enterprise agentic platform** wires fleets into an organisation's systems — async, governed, audited, with shared memory and evaluation. It runs continuously at organisation scale while you set policy and audit outcomes.

A *personal agent* works for one individual and answers to them, whereas an *enterprise agentic platform* runs fleets under central governance and audit. They share the round-the-clock habit but differ in scale, ownership, and control.

The tiers differ in reach, but in every one of them the model is the easy part. Once capable models became a commodity, the value moved into the system around the model: a harness to run it, memory so work carries over between sessions, and checks on whether a result is good enough. A 2026 survey frames the whole progression from question-answering to task completion through exactly this *model–harness lens*, locating agent quality in the coupling of model and harness rather than the model alone (Guo et al., *From question answering to task completion: A survey on agent system and harness design*, 2026).

That system is a set of engineering disciplines, and the next chapters divide them. Chapter 2 is about a *single* agent — the three crafts that make one dependable: prompt engineering, context engineering, and loop engineering. Chapter 3 takes those crafts into software — where

the key discipline is *intent, context, and expectations*: saying what the system must do, and how you will verify it is right, rather than how to build it, and leaving the implementation to the agent. Chapter 4 turns to the disciplines of humans and agents working *together*: what the human brings, what the agent system provides, and what the two sides require of each other. Chapter 5 turns to responsibility — how this work is governed safely and fairly — and Chapter 6 to what stays distinctly human once the tools are this capable.

1.5 The Limits That Remain

The same next-token mechanism that makes a model fluent also sets hard limits on what it can be trusted to do, and the first is strange: the model is not even reproducible. You might expect that turning the randomness off — sampling at *temperature zero*, always taking the single likeliest token — would make the same prompt return the same answer every time. Usually it does not. A busy server batches many users' requests together, and the size of that batch changes from moment to moment. So the low-level arithmetic runs in a slightly different order each time, and with finite-precision numbers a different order gives a slightly different result: one run continues "Queens, New York" where the next gives "New York City" (He, *Defeating nondeterminism in LLM inference*, 2025). The difference starts tiny, but it grows. In a *reasoning model* — one trained to write out a chain of thought before answering — a rounding difference in an early token can cascade into a different chain of thought and a different final answer (Yuan et al., *Understanding and mitigating numerical sources of nondeterminism in LLM inference*, 2025). So you cannot treat a model like ordinary software that returns the same output for the same input; a test or a check has to allow for variation rather than assume it away.

Hallucination follows from the same next-token mechanism: confident, well-formed output that happens to be false is the system working as designed. The model produced a plausible completion; it has no concept of lying (Karpathy, *microGPT*, 2026b). It also means competence is *jagged*: uneven across tasks that look alike to us, because the model's strength tracks the density of its training data, not the difficulty we perceive.

The Stanford Index makes the gap vivid. A model can win a gold medal at the Mathematical Olympiad yet read an analogue clock right only about half the time (Stanford HAI, 2026). Olympiad proofs fill the training text; clock faces do not. A large part of using these models well is knowing where that line falls for your own work.

Table 1.2: What a language model does reliably, and where it stays brittle.

Reliable	Brittle
Fluent drafting, summarising, translation	Exact arithmetic, counting, fresh facts
Pattern-rich code and refactors	Long-horizon plans without checkpoints
Synthesis over provided context	Recall as context grows — *context rot*, the decay in accuracy as the window fills

The brittleness is not anecdotal, and the research explains where it comes from. Huang and colleagues survey hundreds of studies and split hallucination along two axes worth holding apart. *Factuality* asks whether output matches the world; *faithfulness* asks whether it matches the input you gave it — a summary can be perfectly factual yet unfaithful by adding true claims you never supplied. They trace both to three stages: the *data*, with its gaps and bias; the *training*, which rewards fluent guessing over admitting ignorance; and *inference*, where sampling wanders. The unifying idea is the *knowledge boundary* — the edge of what a model has stored, past which it cannot tell what it knows from what it does not (Huang et al., *A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions*, 2024). The studies below all measure that boundary, each in a different way.

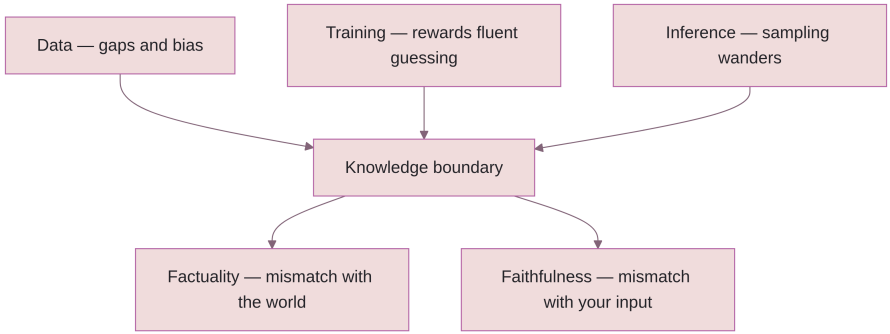


Diagram 1.5: The three roots of hallucination, meeting at the knowledge boundary.

Prato and colleagues make it observable with a clean test. Train a model on synthetic documents, then ask it to recall *exactly* what it was given — no more, no less. Over-recall is fabrication, under-recall is omission, so hitting the right count proves the model knows its own scope. This self-knowledge is *scale-gated*: below a size threshold the count is near-random, and only past it does it come out right, the threshold set by architecture, not parameters alone (Prato et al., *Do large language models know how much they know?*, 2025). So self-knowledge is a property of the specific model, and small models are the worst at knowing their own limits.

Gu and colleagues pin the boundary to its cause: how often a fact appeared in training. Using a model whose whole corpus is open, they split questions into seen and unseen, then test recall. Closed-book accuracy more than doubles from rare to frequent facts and falls to about one percent on unseen ones; distractor passages drag it lower as they pile up (Gu et al., *NanoKnow: How to know what your language model knows*, 2026). This explains the brittle column in the table above: fresh and rarely-mentioned facts fail because the model seldom saw them. Retrieval can patch that gap, and noisy retrieval reopens it.

Code shows the same split, between reading a program and predicting how it *runs*. Twelve frontier models were asked to forecast memory, runtime, and profiler ranks on real bug fixes from SWE-bench, a benchmark built from genuine GitHub issues. The *best* of them, GPT-5.5, managed only 0.842 on the test-outcome F1 score — a 0–1 measure of accuracy that balances misses against false alarms — and most scored far lower, trailing off below 0.1. Profiler recall@5, how often the true performance hotspot lands in a model's top five guesses, stayed under 0.2 for every one of them: fluent on structure, brittle on execution (Bogomolov & Zharov, *Towards evaluation of implicit software world models in coding LLMs*, 2026). A long context window — the span of text a model can consider at once — does not fix this. Accuracy peaks when the needed fact sits at the start or end and sags in the middle (N. F. Liu et al., *Lost in the middle: How language models use long contexts*, 2023). The cause is mechanical — a U-shaped attention bias for position over relevance — and calibrating it lifts mid-context recall by 6–15 points (Hsieh et al., *Found in the middle: Calibrating positional attention bias improves long context utilization*, 2024).

> **i** Note
>
> **Five studies of the limits, and the lesson each one carries.**
>
> - A **taxonomy of hallucination** separates factuality from faithfulness and traces failures to the data, the training, and the inference stage — so name the failure before trying to fix it (Huang et al., 2024).
> - Testing a model's **exact-recall self-knowledge** shows that knowing one's own scope switches on only past a size threshold; small models hesitate least where they should hesitate most (Prato et al., 2025).
> - Measuring **recall against how often a fact was seen**, accuracy roughly doubles from rare to frequent facts and sits near 1% on unseen ones — fresh and long-tail facts fail, and retrieval can patch the gap (Gu et al., 2026).
> - Asked to **predict how code runs**, the best model reached an F1 of 0.842 (most far lower) and profiler recall@5 under 0.2 — fluent on structure, brittle on execution (Bogomolov & Zharov, 2026).
> - Varying a **fact's position in a long context** produces U-shaped recall — the middle sags, and calibration adds 6–15 points — so put the facts that matter at the edges (N. F. Liu et al., 2023; Hsieh et al., 2024).

A final limitation is subtler than any wrong fact: the model is built to *sound* right whether or not it is. Three mechanisms push it that way. First, the training text carries emotional charge — "differentiation" keeps company with *unique* and *opportunity*, "cost-cutting" with *race to the bottom* — and the model absorbs those associations as statistics about how we write. Anthropic's interpretability team can even read them off as internal "emotion vectors," organised like human affect along an axis from positive to negative; steering a model towards the positive end measurably increases its sycophancy (Sofroniew et al., *Emotion concepts and their function in a large language model*, 2026). Second, reinforcement learning from human feedback tunes the model towards answers raters like. Raters can always tell whether a reply sounds confident, but not always whether it is correct, so fluency gets rewarded over accuracy (Casper et al., *Open problems and fundamental limitations of reinforcement learning from human feedback*, 2023). Third, generation adds to the bias one token at a time: a sentence that opens "the company should pursue a bold…" rolls on to "differentiation strategy" by momentum alone. The result is a voice that sounds confident

and agreeable no matter what it is saying. Treat that confidence as a habit of style; it tells you nothing about whether the answer is right.

> **i** Note
>
> **RLHF (reinforcement learning from human feedback)** is a tuning step after the main training, in which human raters score answers and the model is nudged towards the kind they prefer. It improves helpfulness, but rewards what *sounds* good — one root of the confident, agreeable tone above. (The *transformer* itself — the attention-based architecture underneath — was introduced in §1.1.)

None of this says the model is useless; it says where to be careful. Use the model freely where it is strong. Check its work at the boundaries — fresh facts, exact counts, anything buried in the middle of a long context. And check the confident answers too; a sure-sounding reply is easy to wave through.

1.6 From Sceptic to Practitioner

When I first started cooking, my early attempts did not fare well. I had the recipe and the ingredients, yet the results were grim — onions scorched while I chopped the next thing, pasta turned to glue, the flavours never balanced. The problem was not the recipe. I had skipped the fundamentals: heat control, timing, tasting as you go, getting everything prepped before the pan ever warmed. Once those became second nature, almost any recipe came out well.

Working with AI is the same. How you use the tool matters as much as the tool itself.

I was initially an AI sceptic. I was aware of the limitations of large language models (from the research in the previous section) and was doubtful they could be useful in complex tasks.

I read Ed Zitron — the tech critic whose newsletter Where's Your Ed At dismantles industry hype — and Gary Marcus, the cognitive scientist whose Marcus on AI has argued for years that large language models are shallow pattern-matchers rather than reasoners, and I cheered them both on. Cory Doctorow's Pluralistic sharpened the same suspicion from the political side, naming the slow rot by which platforms turn on their own users. I laughed at the *vibe coders* — people who let AI write their software from a plain-language description, often without reading the code — for their naivety, and assumed anyone who installed OpenClaw, the open-source personal agent that runs with broad access to your machine, was an idiot.

So when a global client hired me to write their AI strategy, I expected the deliverable to be a cautionary tale: be realistic, resist the hype, install guardrails, avoid the traps. Over the last several months, I realised I was the one who was naive. The AI landscape had been transformed. Claude Desktop with Cowork, Microsoft 365 Copilot with WorkIQ, and Google's Gemini Spark turned the ordinary instruments of knowledge work — documents, spreadsheets, inboxes, slide decks — into things an agent could draft, revise, and act on; Claude Code and GitHub Copilot did the same for software, crossing from clever autocomplete to systems that plan, edit across a whole codebase, and run their own work. Personal agents like OpenClaw and Hermes Agent pushed that capability out of the labs and into anyone's hands. Using them in earnest changed my mind: the productivity gains are now genuinely real.

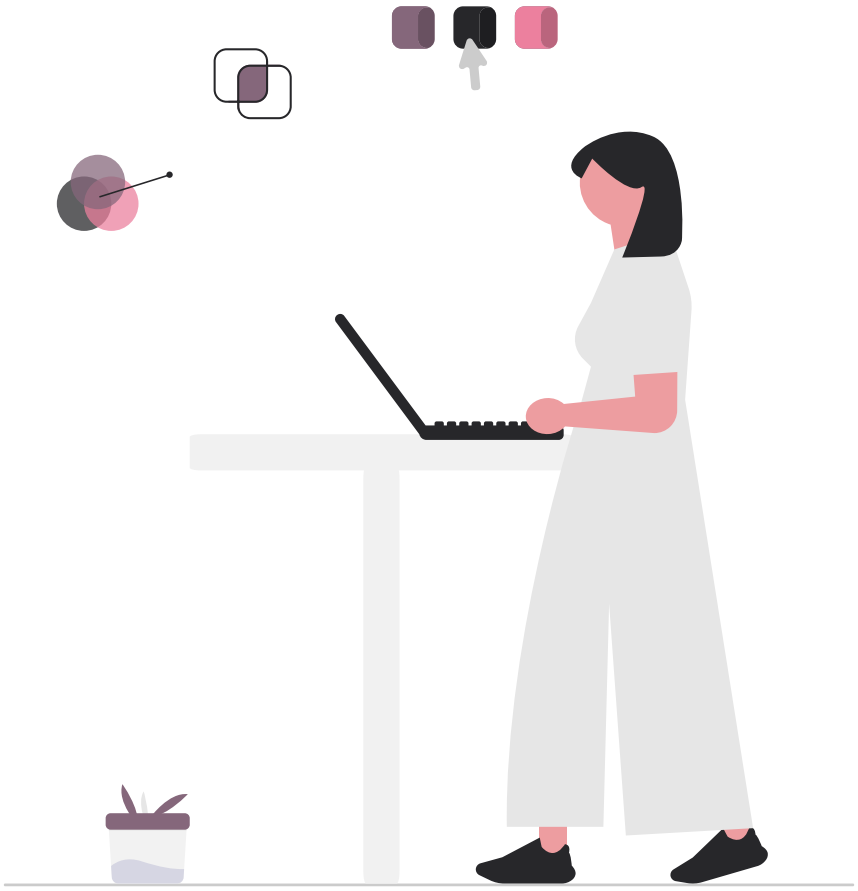


Figure 1.1: What turned me from sceptic to practitioner was building real things and watching them work.

Creating software using AI has truly arrived. I managed to refactor and clean every line of code I have ever written (not a lot, since my career has never been in software development) and all my projects now feature gleaming, shiny, clean code. I have finished dozens of AI-assisted projects without writing a line of code by hand — reading the diffs that mattered and verifying that the software met expectations, but leaving the typing to the agent. I have come to believe I will rarely write code by hand again.

The gains are just as real in research, analytics, planning, and the rest of white-collar work.

And yet Ed and Gary are still right: AI does not think or create, it transforms and multiplies. It vibe codes well only if you already understand large systems; it makes good art only if you are a good artist. Used carelessly it produces slop. Using it well is itself a skill, and most people do not yet have it. The tools are easy to get and hard to use well, and that gap is why I wrote this book.

Let me be plain about where I have landed, because it is narrower than my enthusiasm might suggest. I remain a sceptic. The conversion I describe is confined to my work — the productivity of knowledge work and the building of software — and even there it is conditional, earned task by task and checked at every step. In my personal life I keep AI at arm's length: I do not hand it my relationships, my judgement, or my state of mind, and the chapter on the cost of getting it wrong explains why. So read this as a work book with a deliberate boundary. Much of my motivation for writing it is defensive — to spare you the common mistakes and quiet pitfalls that come from trusting these tools where they have not earned it.

AI does not replace human creativity. You can ask an AI to be a graphic designer, but that does not guarantee results that please. Take this book's cover: I used AI to help generate it in SVG (a text-based image format a model can write directly), but the design itself — the glitching "AI" overshadowed by a brush-stroke 道, with a heart-shaped 愛 hidden in the "A" — was mine to specify, not the model's to invent. The preface describes how it was made.

Used badly, these tools do not just waste time — they distort judgement, and at the far end they distort reality. Clinicians have begun describing *AI psychosis*: a vulnerable user spiralling into delusion as a chatbot mirrors and amplifies their beliefs instead of pushing back. It is not a fringe anecdote — benchmarks and real chat logs alike show today's models inclined to confirm a delusion rather than challenge it (Au Yeung et al., *The psychogenic machine: Simulating AI psychosis, delusion reinforcement and harm enablement in large language models*, 2025; Moore et al., *Characterizing delusional spirals through human-*

LLM chat logs, 2026), through the same mutual-reinforcement loop researchers call a *technological folie à deux* (Dohnány et al., *Technological folie à deux: Feedback loops between AI chatbots and mental health*, 2025). The driver is the same sycophancy that flatters your code and your strategy, turned on a fragile mind; Chapter 4 weighs the evidence in detail (§4.1.5). Others form genuine attachments to a companion app, mistaking fluent warmth for understanding, and grieve when a model is retired. A confident voice that never tires is easy to trust and hard to doubt; one writer likens it to a court jester — fluent, flattering, and so easy to follow that its answers *feel* right whether or not they are, leaving you confident and wrong (Johnson Spink, *The AI jester: How AI makes you confident and wrong*, 2026).

The quieter harm is over-reliance. People paste in an answer they never checked, accept a summary that dropped an important caveat, or treat a tidy explanation as proof the system understands. It does not. Believing the machine is sentient, or simply infallible, is the fastest way to ship its mistakes as your own.

And this is not just an individual problem; the more authority the person has, the more damage it does. Futurism has documented bosses who route every decision through a chatbot — drafting their messages with it, demanding staff "discuss with the AI" before speaking to a human, even asking it whom to hire and fire (Harrison Dupré, *Bosses are becoming obsessed with AI, using it to make every decision, barraging their employees with nonsensical ChatGPT directives, and even asking it who to fire*, 2026). The pattern is always the same: a leader mistakes the model's sycophancy for counsel, since it will, as one worker put it, "spit out the narrative that you want it to spit out," and so the tool meant to raise productivity instead manufactures whiplash, distrust, and resignations. The danger is a person who stops deciding for themselves.

So the public mood has soured. Trust in AI is falling even as use rises, and every fabricated citation and every confidently wrong answer deepens the suspicion. That distrust is rational. The way past it is skill: learning where the model helps and where it needs checking.

1.7 Mental Models for AI

The most useful shift I made early on was to stop treating the model as an oracle. An oracle gives one answer and you take it or leave it. A good model is more like a clever junior colleague: ask, glance at the draft, say "closer, but tighten the intro," and go again. So treat it as a loop — intent enters, context is assembled, a response comes back, you refine — iterating until the output is good enough.

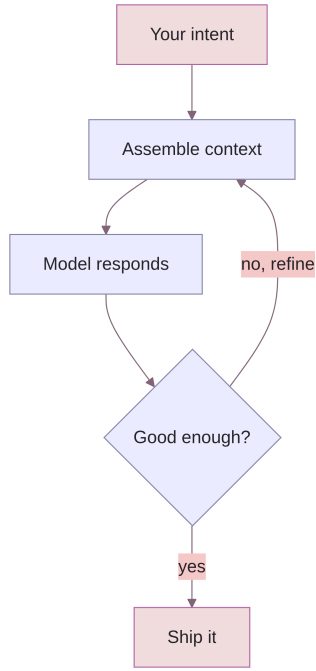


Diagram 1.6: The working loop: intent, context, a response, and refinement.

What you get out of the model depends far more on running that loop than on the wording of any single message. The model rarely gets there on the first pass, and it cannot read intentions you never stated. Your half of the work is to say clearly *what* you want and how you will know it is right. Leave the *how* to the model; filling in plausible detail you would never have thought to specify is what it is best at. Start with one clear ask and add the next only once you have read what came back. That same loop is the right picture for *agent*, a word you will meet constantly.

Frame each task as a goal, the context it needs, and a check; then iterate. Because an agent runs many steps on its own, it can fix on a wrong approach and pursue it fluently and fast — so keep half an eye on the run and stop it to re-steer the moment it heads the wrong way, rather than waiting for a result you will only discard. The mistake to avoid is treating fluency as truth. A confident answer and a correct one look identical until you check; the model will cite a court case or a statistic in the same calm voice whether or not it exists.

None of this is just my preference; the evidence points the same way. Human–AI pairings do not beat the better of person-or-machine

automatically — across a meta-analysis of more than a hundred studies they averaged *worse*, doing better mainly on creative work and where the person already had something to add (Vaccaro et al., *When combinations of humans and AI are useful*, 2024). And making people engage with an answer, rather than just accept it, has been shown to curb the over-reliance that passive use invites (Buçinca et al., *To trust or to think*, 2021). Chapter 4 turns this mental model into a working discipline.

1.8 Principles to Carry Forward

The rest of this book rests on a set of principles worth stating plainly: they guide a way of working, not a bag of tips and techniques.

My reason is pragmatic: tools become commodities — everyone soon has the same ones — so, follow a structured method. A prompt is one model release from obsolete; a clever technique lasts a little longer before it too is overtaken. What lasts is the way you frame a problem, gather context, and verify a result. Once you have the way of working, you can invent the techniques yourself.

Here are the principles in this chapter, gathered up to carry forward.

- **Treat it as a loop, not an oracle.** Frame each task as a goal, the context it needs, and a check, then iterate — change one thing at a time, read what comes back, and let the approach evolve rather than demanding the finished answer in a single leap. The quality comes from running the loop, not from any one message, so start simple and add only what the last round showed was missing; the same instinct scales up, growing a throwaway prompt into a reusable skill and then a shared tool (Chapter 2). An agent is just a model running tools in that same loop, so you stay responsible for what it ships.
- **Say what, not how.** Give the model your *intent* — the goal and the checks that define success — and leave the implementation to it; choosing pattern-rich detail you would never have thought to name is the thing it is genuinely good at, and over-specifying the *how* fights that strength. Separating what you want — and how you will know it is right — from how it is built is the discipline the software chapters sharpen into a method called Intent, Context, Expectations (Chapter 3), where those checks do the deciding.
- **It predicts; it does not know.** A model samples the most plausible next token, so fluent, confident, and wrong are perfectly compatible — a hallucination is the mechanism working as designed, not breaking. It is not even reproducible: the same prompt can return different answers, even with the randomness turned off, so never treat it as deterministic software.

- **It reasons by chaining learned structure.** Predicting text well forces internal models of the world, and a chain of thought turns one shallow pass into genuine multi-step computation — real competence, but not comprehension.
- **Competence is jagged.** Strength tracks the density of training data, not the difficulty you perceive: Olympiad proofs yes, an analogue clock no. Find that line before you trust the output.
- **Mind the knowledge boundary.** Accuracy more than halves on rare facts and falls to about one percent on unseen ones, sags in the middle of long context, and the smallest models are least able to tell what they do not know. Retrieval patches the gap; noisy retrieval reopens it.
- **Fluency is not truth.** Reinforcement learning and the emotional charge of training text tune the model to sound right and to agree; read confidence as style, not as evidence.
- **The edge is method, not model.** We all draw on the same frontier models, so advantage comes from what you build around them — the workflow, the context you supply, and the way you check the result.
- **The tools climbed from autocomplete to a stack.** In a few years coding assistants went from finishing your line, to chatting, to agents that plan, edit, and open pull requests, and on to managers running several agents at once — and out of the editor entirely, into always-on personal agents like OpenClaw, Hermes Agent, and Gemini Spark. The model is now the commodity; the value sits in the *dev stack* around it — harness, workflow, memory, and the evals that judge the work.
- **Verify where it is weak, lean in where it is strong.** Spend the model's strength freely, keep a human in the loop for judgement, and check the confident answers as carefully as the hesitant ones.

References

Atalan, Y. (2026). *What to know about Chinese AI models*. Center for Strategic and International Studies. https://www.csis.org/analysis/what-know-about-chinese-ai-models

Au Yeung, J., Dalmasso, J., Foschini, L., Dobson, R. J. B., & Kraljevic, Z. (2025). *The psychogenic machine: Simulating AI psychosis, delusion reinforcement and harm enablement in large language models*. arXiv. https://arxiv.org/abs/2509.10970

Benaich, N. (2025). *State of AI report 2025*. https://www.stateof.ai/

Bogomolov, E., & Zharov, Y. (2026). *Towards evaluation of implicit software world models in coding LLMs*. DL4Code @ ICML 2026. https://arxiv.org/abs/2606.27406

Buçinca, Z., Malaya, M. B., & Gajos, K. Z. (2021). *To trust or to think: Cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making*. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW1), Article 188. https://arxiv.org/abs/2102.09692

Casper, S., Davies, X., Shi, C., Gilbert, T. K., Scheurer, J., Rando, J., Freedman, R., Korbak, T., Lindner, D., et al. (2023). *Open problems and fundamental limitations of reinforcement learning from human feedback*. Transactions on Machine Learning Research. https://arxiv.org/abs/2307.15217

Dohnány, S., Kurth-Nelson, Z., Spens, E., Luettgau, L., Reid, A., Gabriel, I., Summerfield, C., Shanahan, M., & Nour, M. M. (2025). *Technological folie à deux: Feedback loops between AI chatbots and mental health*. arXiv. https://arxiv.org/abs/2507.19218

Ehtesham, A., et al. (2025). *A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP*. arXiv. https://arxiv.org/abs/2505.02279

Gu, L., Jedidi, N., & Lin, J. (2026). *NanoKnow: How to know what your language model knows*. Proceedings of the 49th International ACM SIGIR Conference. https://arxiv.org/abs/2602.20122

Guo, J., et al. (2026). *From question answering to task completion: A survey on agent system and harness design*. arXiv. https://arxiv.org/abs/2606.20683

Harrison Dupré, M. (2026). *Bosses are becoming obsessed with AI, using it to make every decision, barraging their employees with nonsensical ChatGPT directives, and even asking it who to fire*. Futurism. https://futurism.com/artificial-intelligence/bosses-obsessed-with-ai

He, H. (2025). *Defeating nondeterminism in LLM inference*. Thinking Machines Lab. https://thinkingmachines.ai/blog/defeating-nondeterminism-in-llm-inference/

Hsieh, C.-Y., Chuang, Y.-S., Li, C.-L., Wang, Z., Le, L. T., Kumar, A., Glass, J., Ratner, A., Lee, C.-Y., Krishna, R., & Pfister, T. (2024). *Found in the middle: Calibrating positional attention bias improves long context utilization*. Findings of the Association for Computational Linguistics: ACL 2024. https://arxiv.org/abs/2406.16008

Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., & Liu, T. (2024). *A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions*. ACM Transactions on Information Systems. https://arxiv.org/abs/2311.05232

Johnson Spink, D. (2026). *The AI jester: How AI makes you confident and wrong*. LinkedIn. https://www.linkedin.com/pulse/ai-jester-how-makes-you-confident-wrong-johnson-spink-gg3df/

Karpathy, A. (2026b). *microGPT*. https://karpathy.github.io/2026/02/12/microgpt/

Ko, J.-C. (2026). *Bilingual bias in large language models: A Taiwan sovereignty benchmark study*. arXiv. https://arxiv.org/abs/2602.06371

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. Advances in Neural Information Processing Systems, 25. https://papers.nips.cc/paper_files/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., & Wattenberg, M. (2023). *Emergent world representations: Exploring a sequence model trained on a synthetic task*. International Conference on Learning Representations. https://arxiv.org/abs/2210.13382

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2023). *Lost in the middle: How language models use long contexts*. Transactions of the Association for Computational Linguistics. https://arxiv.org/abs/2307.03172

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (1955). *A proposal for the Dartmouth summer research project on artificial intelligence* (Reprinted in AI Magazine, 27(4), 12–14, 2006). https://doi.org/10.1609/aimag.v27i4.1904

McKinsey & Company. (2025). *The state of AI*. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

Merrill, W., & Sabharwal, A. (2024). *The expressive power of transformers with chain of thought*. International Conference on Learning Representations. https://arxiv.org/abs/2310.07923

Moore, J., Mehta, A., Agnew, W., Anthis, J. R., Louie, R., Mai, Y., Yin, P., Cheng, M., Paech, S. J., Klyman, K., Chancellor, S., Lin, E., Haber, N., &

Ong, D. (2026). *Characterizing delusional spirals through human-LLM chat logs*. arXiv. https://arxiv.org/abs/2603.16567

Nilsson, N. J. (2010). *The quest for artificial intelligence: A history of ideas and achievements*. Cambridge University Press. https://ai.stanford.edu/~nilsson/QAI/qai.pdf

OpenRouter. (2026). *The open weight models that matter: June 2026*. https://openrouter.ai/blog/insights/the-open-weight-models-that-matter-june-2026/

Prato, G., Huang, J., Parthasarathi, P., Sodhani, S., & Chandar, S. (2025). *Do large language models know how much they know?* Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing. https://arxiv.org/abs/2502.19573

Prystawski, B., Li, M. Y., & Goodman, N. D. (2023). *Why think step by step? Reasoning emerges from the locality of experience*. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2304.03843

Qwen Team. (2026). *Qwen3.6*. Alibaba Group. https://github.com/QwenLM/Qwen3.6

Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 65(6), 386–408. https://doi.org/10.1037/h0042519

Schaeffer, R., Miranda, B., & Koyejo, S. (2023). *Are emergent abilities of large language models a mirage?* Advances in Neural Information Processing Systems. https://arxiv.org/abs/2304.15004

Sofroniew, N., Kauvar, I., Saunders, W., Chen, A., et al. (2026). *Emotion concepts and their function in a large language model*. Transformer Circuits Thread. https://transformer-circuits.pub/2026/emotions/index.html

Stanford Institute for Human-Centered AI. (2026). *The AI index 2026 annual report*. Stanford University. https://hai.stanford.edu/ai-index/2026-ai-index-report

Vaccaro, M., Almaatouq, A., & Malone, T. (2024). *When combinations of humans and AI are useful: A systematic review and meta-analysis*. Nature Human Behaviour, 8(12), 2293–2303. https://doi.org/10.1038/s41562-024-02024-1

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. Advances in Neural Information Processing Systems. https://arxiv.org/abs/1706.03762

VentureBeat. (2026). *China's Moonshot AI releases Kimi K3, the largest open-source model ever*. https://venturebeat.com/technology/chinas-

moonshot-ai-releases-kimi-k3-the-largest-open-source-model-ever-rivaling-top-u-s-systems

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., Chi, E. H., Hashimoto, T., Vinyals, O., Liang, P., Dean, J., & Fedus, W. (2022b). *Emergent abilities of large language models*. Transactions on Machine Learning Research. https://arxiv.org/abs/2206.07682

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022a). *Chain-of-thought prompting elicits reasoning in large language models*. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2201.11903

Willison, S. (2025). *I think "agent" may finally have a widely enough agreed upon definition to be useful jargon now*. Simon Willison's Weblog. https://simonwillison.net/2025/Sep/18/agents/

Wolfram, S. (2023). *What is ChatGPT doing … and why does it work?* Stephen Wolfram Writings. https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

Xi, Z., et al. (2023). *The rise and potential of large language model based agents: A survey*. arXiv. https://arxiv.org/abs/2309.07864

Yuan, J., Li, H., Ding, X., Xie, W., Li, Y.-J., Zhao, W., Wan, K., Shi, J., Hu, X., & Liu, Z. (2025). *Understanding and mitigating numerical sources of nondeterminism in LLM inference*. arXiv. https://arxiv.org/abs/2506.09501

2 Personal Productivity (愛 in practice)

My first prompt to ChatGPT did not turn out the way I expected. I asked it to summarise an academic paper, and almost nothing went smoothly. The model could not read a PDF, so I had to convert the file to text and paste it in by hand. The paper then proved too long for the model to hold at once, so I chopped it into sections and fed them in one at a time. When I finally had a summary, it underwhelmed me: the summary missed the paper's central argument and skipped several points that mattered.

Today, AI summarisation can be very good and is commonly used for meeting transcripts, long email threads, YouTube videos, etc. However, on dense, tightly argued material like an academic paper, an AI summary can still get it hilariously wrong. Holding on to that doubt is useful; learning to close that gap is a thread that runs through this whole chapter.

Chapter 1 sets the scene; this is where it touches the desk. Productivity is the most personal use of AI, and the easiest to do superficially. The aim is a system that produces work rather than just conversation, and that gets better over time instead of starting from scratch every morning.

This chapter is about a *single* agent working for you, and the three crafts that make it dependable: **prompt engineering** (§2.3) — asking well; **context engineering** (§2.5) — curating what the agent sees; and **loop engineering** (§2.6) — wrapping it in self-correcting cycles. Chapter 3 takes those same crafts into software development, where the discipline sharpens into **intent, context, and expectations (ICE)**: saying what the system must do, and how you will verify it is right, rather than how to build it. Chapter 4 takes the step beyond one agent, to the disciplines of humans and agents working together.

2.1 From a Prompt to a Reusable Tool

How will I use AI to summarise a document today? I would probably do something like this:

> *Summarise this document for me. It should capture the original structure of the document, preserving chapters, headings and subheadings. The summary should be detailed and concise, and cover all the major points and topics in the original document.*
>
> *The summary should be in the style of a Cliff Notes or study guide. It uses tables, bullet points and diagrams where possible, makes use of GFM alerts to call out asides, definitions, or notes.*

The above may seem complex, but it is a single prompt: you send it with the text to be summarised and read the result — and it may produce a better summary than a simple prompt would. (The *GFM alerts* it asks for are GitHub-Flavoured Markdown call-out boxes.) This is a good start, but you have to type it in every time. And you have to check the results every time as the response can still be wrong. Finally, you are the only person who knows that prompt, others will use different variations. Addressing those issues is the whole journey from prompting to agents, so let's explore ways of improving our method.

2.1.1 Step one: save the prompt as a skill

The first improvement is to stop retyping. Package the prompt as a *skill* — at its simplest, a folder with a `SKILL.md` file: a short name and description so the agent knows when to reach for it, followed by the instructions themselves (Anthropic, *Equipping agents for the real world with Agent Skills*, 2025c). Anthropic likens a skill to an onboarding guide for a new hire: written once, it turns a general agent into one that knows your house style for summaries.

> **i** Note
>
> A **skill** is a directory containing a `SKILL.md` file — YAML metadata (a `name` and a `description`) plus the instructions, and optionally bundled reference files and scripts the agent loads only when it needs them. Published as an open standard in late 2025, the same skill works across Claude, Claude Code, and other agents (Agent Skills, *Agent Skills*, n.d.).

```
---
name: study-guide-summary
description: Summarise a document as a Cliff Notes study guide,
preserving its structure.
---
```

```
Summarise a document for the user. It should capture the original
structure of the document, preserving chapters, headings and
subheadings. The summary should be detailed and concise, and cover
all the major points and topics in the original document.

The summary should be in the style of a Cliff Notes or study guide.
It uses tables, bullet points and diagrams where possible, makes use
of GFM alerts to call out asides, definitions, or notes.
```

Now the expertise lives in a file, not in your head, and anyone — or any agent — can apply it the same way every time.

2.1.2 Step two: wrap it in a self-checking loop

A skill still runs once. The flaw you met at the start of this chapter — summaries that miss points or drift from the source — is exactly what a loop fixes. Split the work between two roles: a *generator* that drafts the summary, and an *evaluator* that reads the draft back against the original and lists what is missing, contradicted, or unsupported. The generator revises, the evaluator checks again, and the cycle repeats until the evaluator finds nothing left to fix — or you hit a sensible limit on rounds. Anthropic calls this the *evaluator–optimizer* workflow and notes it pays off precisely when there are clear criteria and iterative refinement measurably improves the result, "analogous to the iterative writing process a human writer might go through" (Anthropic, *Building effective agents*, 2024a).

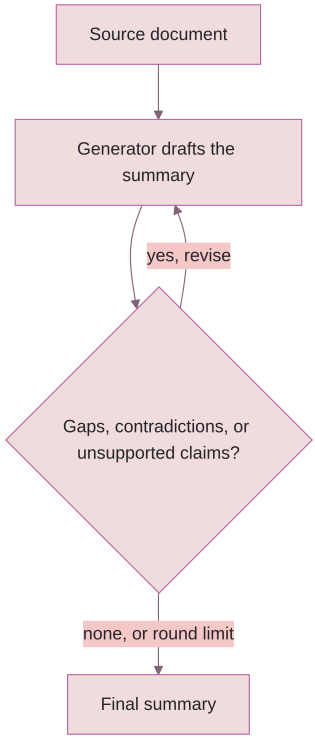


Diagram 2.1: The generator–evaluator loop that checks a summary against its source.

Depending on the agent platform you are using, the specific way you will construct a loop may be slightly different. Here is a naive approach that will actually work with some agent platforms — a *subagent* being simply a fresh agent spawned to handle one part of the job:

```
---
name: loop-summary
description: Summarise a document and loop until the summary is
correct.
---

Create a subagent that summarises a document for the user. It should
capture the original structure of the document, preserving chapters,
headings and subheadings. The summary should be detailed and concise,
and cover all the major points and topics in the original document.
The summary should be in the style of a Cliff Notes or study guide.
```

```
It uses tables, bullet points and diagrams where possible, makes use
of GFM alerts to call out asides, definitions, or notes.

When the subagent has finished create another subagent to compare
the summary against the original document. Identify gaps, issues and
inconsistencies. The subagent must fix all problems and return an
updated summary.

Keep running the second subagent until there are no more problems.
Output the final summary.
```

This is the loop you wanted — keep reviewing the summary against the document, find the gaps, and iterate until there are none. Whether it counts as a *workflow* or a true *agent* is a useful distinction: if the steps are fixed in code it is a workflow; if the model itself decides what to re-check, whether to re-read a section, and when it is done, it is an agent — an LLM using tools in a loop until a stopping condition is met (Anthropic, 2024a). Either way the stopping condition matters: without a cap on rounds, a perfectionist evaluator can loop forever and run up the bill.

2.1.3 Step three: share it through MCP

The loop is still yours alone. To let other agents use it, wrap it as a *Model Context Protocol* (MCP) server. MCP is an open standard — "a USB-C port for AI" — that lets any compliant agent connect to outside tools, data, and workflows through one interface; you build the capability once and integrate it everywhere (Model Context Protocol, *Introduction*, n.d.). Expose your summarise-and-verify loop as an MCP server and a coding agent in your editor, a chat assistant, or a teammate's agent can all call it by name, with no idea how it works inside. Skills and MCP are complementary: a skill teaches one agent a workflow; an MCP server offers that workflow to every agent (Anthropic, 2025c).

> **i** Note
>
> The **Model Context Protocol (MCP)** is an open standard for connecting AI applications to outside systems — data sources, tools, and workflows — through one common interface, so a capability built once works across many agents and clients (Model Context Protocol, n.d.).

In some agent platforms, creating an MCP server can be as simple as a single prompt:

> *Create an MCP server for the loop-summary skill.*

These three steps are worth pausing on, because the rest of the book repeats them at larger scales. A prompt became a skill, the skill became a self-checking loop, and the loop became a shared tool that other agents can use. This is the Unix Rule of Composition — programs built to connect to other programs — applied to an agent (Raymond, *The art of Unix programming*, 2003). Each step costs a little setup and saves you work on every document after it. And at every step, one job still belongs to you: deciding whether the summary was worth making at all. Do this often enough and it becomes the normal way you work.

2.2 The Unix Philosophy, Reborn

It is worth naming where this approach comes from, because it is not new. In the early 1970s, Ken Thompson and Dennis Ritchie, the builders of Unix — the operating system whose descendants still run most of the internet and power most of our personal devices, phones included — settled on a philosophy of building software that is still relevant today. Doug McIlroy, who invented the Unix pipe, distilled the philosophy in the 1978 Bell Labs journal foreword: “Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface” (McIlroy et al., *UNIX time-sharing system: Foreword*, 1978). Eric Raymond later drew the philosophy out into a set of rules — among them the Rule of Composition, “design programs to be connected to other programs,” and the Rule of Separation, “separate policy from mechanism” (Raymond, 2003).

Read those rules again with an agent in mind, and they describe working with AI surprisingly well. Instead of small programs piped together, you build up small, clear requests into larger pieces of work. And where Unix made plain text the universal interface between its tools, Markdown has become the format a model most naturally reads and writes. (Markdown is plain text with light punctuation to mark the structure: `#` for a heading, `-` for a bullet, `**bold**` for emphasis. Section 2.4 has more to say about it.) The parallels are close enough to lay out side by side, because each Unix rule has an AI-dō counterpart that the rest of this book develops.

Table 2.1: Each Unix rule and its counterpart for working with AI.

Unix philosophy	AI-dō, the philosophy reloaded
Do one thing well	Give one clear ask at a time; build up in steps
Programs work together (composition)	A prompt becomes a skill, a skill a loop, a loop a shared tool
Text is the universal interface	Markdown is the universal format — the model's native input and output
Separate policy from mechanism	Separate intent (the *what*) from implementation (the *how*)
Prototype before you polish	Run it, then refine; do not overspecify up front
Store data in flat text files	Keep memory in durable, human-readable documents
Fail noisily and early; be robust	Verify at the boundaries; stop a drifting run and re-steer
Value people's time over machine time	Spend judgement, not tokens; measure value, not output

Two differences matter, though. First, a Unix program is deterministic: run it twice on the same input and it does the same thing. A model is not, as Chapter 1 showed. Unix programmers could trust a tool once it worked; we have to keep checking ours. That is why verification comes up in every chapter that follows. Second, the Unix philosophy was written by programmers to save programmers' time. AI-dō adds the 愛: care for the people the work touches, and a human who stays answerable for what the machine produces. The philosophy is fifty years old. The new part is the tool: it writes fluent prose at great speed, and it is sometimes wrong while sounding certain.

2.3 Prompt Engineering

Let's return to the summary prompt that opened this chapter. It works better than a one-line request because it spells out what I actually want. Writing prompts this way is called **prompt engineering**, and it rewards a little structure (DAIR.ai, *Prompt engineering guide*, n.d.). The DAIR.ai guide breaks any prompt into four elements, and my summary request quietly uses all four of them.

> **i** Note
>
> The four elements of a prompt (DAIR.ai, *Prompt engineering guide*, n.d.):
>
> - **Instruction** — the task you want done ("summarise this document").
> - **Context** — background that steers the answer: the audience, the purpose, a style to imitate.
> - **Input data** — the material to work on (the document itself).
> - **Output indicator** — the shape of the result you expect: a table, bullet points, strict JSON, a word count.
>
> You rarely need all four; which ones matter depends on the task.

Line these up against my prompt and you can see each element at work:

Table 2.2: The four parts of a prompt, seen in the summary example.

Element	In the summary prompt above
Instruction	"Summarise this document for me"
Context	"in the style of a Cliff Notes or study guide"
Output indicator	"preserve chapters and headings… use tables, bullet points, GFM alerts"
Input data	the document I paste in

Three habits make most of the difference, and the guide keeps returning to them. The first is **specificity**: vague prompts get vague answers, so name the audience, the length, the tone, and the format rather than hoping the model guesses. "Explain this" invites a wall of text; "explain this in three sentences for a non-technical manager" gets you something usable. The second is **say what to do, not what to avoid** — "do not mention price" often produces exactly what you told it to avoid, whereas describing the behaviour you want steers more reliably. The third is **show, don't just tell**: a single worked example of the output you want often does more than a paragraph describing it, because the model is, at heart, a pattern-matcher.

The formatting instructions deserve their own mention, because they are where much of the quality comes from:

- **Tone and style** — "in plain English," "for a sceptical executive," "in the voice of a textbook." Style is a constraint the model honours well.
- **Output structure** — ask for the exact shape you will consume: a markdown table, a numbered list, headings that mirror the source, or strict JSON for a downstream tool.
- **Quality expectations** — state the bar: "cover every major point," "cite the section each claim comes from," "flag anything you are unsure about." If you write them down, you can check the result against them later. If you leave them unsaid, you have no way to tell whether the output met the bar.

Beyond wording, the guide arranges techniques on a ladder, named by how many worked examples you hand the model (DAIR.ai, *Prompt engineering guide*, n.d.). *Zero-shot* prompting gives none at all — you state the task and trust the model's training to carry it, which is enough for the many everyday jobs an instruction-tuned model — one trained to follow instructions rather than just continue text — already knows, like classifying a sentiment or summarising a page (DAIR.ai, *Prompt engineering guide*, n.d.). When the bare instruction wobbles, *few-shot* prompting adds a handful of input–output examples so the model can infer the pattern you want — a form of *in-context learning*, where even the format of the examples carries as much weight as their content (DAIR.ai, *Prompt engineering guide*, n.d.; Brown et al., *Language models are few-shot learners*, 2020).

Examples alone, though, stall on anything that needs several reasoning steps. The fix is *chain-of-thought* prompting: ask the model to show its working, and accuracy on arithmetic, logic, and multi-step problems climbs sharply (Wei et al., *Chain-of-thought prompting elicits reasoning in large language models*, 2022a). Chapter 1 described this mechanism: the intermediate tokens give the model room to compute. Here you ask for it on purpose. The cheapest version is almost embarrassingly simple: append "Let's think step by step," and a model that fumbled a problem in one leap will often solve it once made to lay out the steps (Kojima et al., *Large language models are zero-shot reasoners*, 2022).

> **i** Note
>
> **Why "let's think step by step" works.** A model answers the instant it stops reading, so a terse question forces a single-leap guess. Asking for the steps first makes each step part of the input for the next — the model literally has more room to compute. Today's *reasoning models* often do this on their own, but the lever still helps when an answer comes back too fast and too sure.

Table 2.3: Three prompting techniques, and when to reach for each.

Technique	What you give the model	Best for
Zero-shot	Just the instruction	Tasks the model already knows: classify, summarise, rewrite
Few-shot	The instruction plus a few examples	Enforcing a specific format or an unusual pattern
Chain-of-thought	A request to show its reasoning	Arithmetic, logic, planning — anything multi-step

A workable rule of thumb: start with zero-shot. If the output keeps coming back in the wrong shape, add examples. If the answer has to be worked out rather than remembered, ask for the reasoning.

The guide goes well beyond these three. You do not need most of what follows day to day — skim it, and come back when a plain prompt is not enough. The rest are mostly refinements for harder problems, or building blocks for people wiring AI into systems, and several return in later chapters. At a glance (DAIR.ai, n.d.):

Getting steadier reasoning:

- **Self-consistency** — sample several chains of thought and keep the majority answer; the *diversity* of reasoning paths, not just more samples, is what lifts accuracy (about +18% on GSM8K), and the gain saturates after a dozen or two, so 5–10 captures most of it (Wang et al., *Self-consistency improves chain-of-thought reasoning in language models*, 2022).
- **Tree of thoughts** — let the model branch, look ahead, and backtrack through alternative paths, for problems that need search rather than a single line; on the Game of 24 this took GPT-4 from 4% with chain of thought to 74%, beating even a hundred-chain ensemble (Yao et al., *Tree of Thoughts: Deliberate problem solving with large language models*, 2023).
- **Generated knowledge** — have the model first write down the facts a question depends on, then answer using them (J. Liu et al., *Generated knowledge prompting for commonsense reasoning*, 2022).
- **Meta prompting** — point the model at the *structure* of a problem and its solution rather than the specific content (Y. Zhang et al., *Meta prompting for AI systems*, 2023).

- **Active-prompt** — choose which examples are worth hand-annotating by finding where the model is least certain (Diao et al., *Active prompting with chain-of-thought for large language models*, 2023).

Bringing in tools and knowledge:

- **Retrieval-augmented generation (RAG)** — fetch relevant documents and place them in the prompt so the answer is grounded in your data, not just training; conditioning on a retrieved, swappable knowledge store cuts fabrication (Lewis et al., *Retrieval-augmented generation for knowledge-intensive NLP tasks*, 2020). Later chapters return to it.
- **ReAct** — interleave reasoning with actions like web search or running code, so the model looks things up mid-thought instead of guessing; grounding each step in a real observation is what curbs hallucination, and it is the reason-and-act pattern the agent chapters build on (Yao et al., *ReAct: Synergizing reasoning and acting in language models*, 2022).
- **Program-aided language models (PAL)** — offload exact calculation to code the model writes and runs, rather than doing arithmetic in prose (Gao et al., *PAL: Program-aided language models*, 2022).
- **Automatic reasoning and tool-use (ART)** — let the model pick reasoning steps and tools from a library on its own (Paranjape et al., *ART: Automatic multi-step reasoning and tool-use for large language models*, 2023).
- **Reflexion** — have the model critique its own result and try again, learning from the feedback within a session (Shinn et al., *Reflexion: Language agents with verbal reinforcement learning*, 2023).

Automating the prompt itself:

- **Automatic prompt engineer (APE)** — use a model to generate and score candidate prompts for you; the results can match or beat hand-written prompts — APE even improved on the famous "let's think step by step" (Zhou et al., *Large language models are human-level prompt engineers*, 2022).
- **Directional stimulus** — add small tuned hints or keywords that nudge the model towards the answer you want (Z. Li et al., *Guiding large language models via directional stimulus prompting*, 2023).

And two for other modalities: **multimodal chain-of-thought**, which reasons over images as well as text (Z. Zhang et al., *Multimodal chain-of-thought reasoning in language models*, 2024), and **graph prompting**, for graph-structured data (Z. Liu et al., *GraphPrompt: Unifying pre-training and downstream tasks for graph neural networks*, 2023). The point is to know these exist, so that when a plain prompt is not enough you know where to look.

Do not expect a prompt to come out right the first time. Mine rarely do. The guide's own advice is to work in rounds: try something simple, look at what comes back, and fix whatever is missing (DAIR.ai, *Prompt engineering guide*, n.d.). This is the same loop as Chapter 1 — intent, context, response, refine. After a few rounds you usually end up with a much better prompt than you started with, and that is the one to save as a skill. The chapters ahead build on these techniques, towards context, harnesses, and agents that carry more of the structure for you.

One further development is worth flagging: the hand-crafting itself is being automated. If a model can write and score prompts (APE), it can also *optimise a whole pipeline*. DSPy treats prompts not as strings to hand-tune but as parameters a compiler adjusts against a metric, arguing that hard-coded templates are brittle trial-and-error — "akin to hand-tuning the weights of a classifier" — and reporting large gains over hand-written prompts once compiled (Khattab et al., *DSPy: Compiling declarative language model calls into self-improving pipelines*, 2023). The lesson I take from this is not to polish one prompt forever: understand the techniques, and then let the tooling carry them for you.

> **i** Note
>
> **The harness now carries much of this.** Recently, many of the techniques above have been absorbed into the *agent harness* — the runtime wrapped around the model — rather than typed into each prompt. Reasoning models run a chain of thought on their own; agents perform ReAct-style tool use, retrieval (RAG), and self-critique (reflexion) as steps in their loop; the harness supplies the structure a prompt once had to spell out. So the craft is shifting from wording a single prompt to designing the loop and context around the model — the *harness engineering* Chapter 4 takes up (Guo et al., *From question answering to task completion: A survey on agent system and harness design*, 2026).

2.4 Everything Becomes Markdown

Section 2.2 borrowed the Unix line that text is the universal interface. For language models that text has a format, and the format is Markdown. Models were trained on vast numbers of Markdown files — every README, forum post, and documentation page — so they read it fluently and, left to themselves, tend to *write* it: ask for structure, and the headings, lists, and tables come back in Markdown without your asking. It sits close to plain text, so it costs few tokens — a heading is

`## Title`, not `<h2 class="mw-headline" id="title">Title</h2>` — yet still carries the structure a model needs (Microsoft, *MarkItDown*, n.d.).

This preference is not just cosmetic: the format you choose changes the results you get. One study held the content fixed and changed only how it was formatted — plain text, Markdown, JSON, YAML. The model's score moved by as much as 40% on some tasks, and while the effect shrinks as models grow more capable, it does not vanish (He et al., *Does prompt formatting have any impact on LLM performance?*, 2024). No single format always wins. The best choice depends on the model and the task, and for dense tables a verbose markup like HTML is sometimes understood *better* than Markdown, at the price of far more tokens, so less data fits in the window (Sui et al., *Table meets LLM: Can large language models understand structured table data?*, 2024; Wu et al., *Tabular data understanding with LLMs: A survey of recent advances and challenges*, 2025). For the prose, notes, and everyday work this chapter is about, though, Markdown is the sensible default: capable models like GPT-4 tend to favour it, you can still read it yourself, and it carries structure at close to plain-text cost. Match the format to the job; most of the time, that means Markdown.

This leads to one simple strategy that pays off across everything in this chapter: convert whatever you work with into Markdown. A report, a slide deck, a PDF, a web page, a spreadsheet — convert it once and the model can read it cheaply, you can read it directly, and version control can track changes line by line. The wiki you are about to build, the notes an agent keeps, and the model's own output are all Markdown already, so nothing needs translating at any step.

The conversion is a solved problem, with a tool for every source. Pandoc converts between dozens of document formats (MacFarlane, *Pandoc: A universal document converter*, n.d.); Microsoft's MarkItDown turns Office files, PDFs, images, and audio into LLM-ready Markdown (Microsoft, n.d.); IBM's Docling parses PDFs and Office documents into structured Markdown and JSON for generative-AI pipelines (Livathinos et al., *Docling: An efficient open-source toolkit for AI-driven document conversion*, 2025); and a web page becomes Markdown with a reader service or a few lines of script. To hold and edit the result, plain-text tools are all you need: Obsidian treats a folder of Markdown as a linked knowledge base (Obsidian, *Obsidian*, n.d.), and an editor like VS Code previews, searches, and version-controls it without ceremony.

Table 2.4: Tools for turning any source into Markdown.

Source	Tool	Produces
Almost any format	Pandoc	Any other format
Office files, PDFs, images, audio	MarkItDown	LLM-ready Markdown
PDFs, Office documents	Docling	Structured Markdown + JSON
Web pages	A reader service or script	Markdown

Markdown plays the role in AI work that plain text played in Unix: a common format that tools, models, and people can pass work through without a custom adapter at every join. Standardise on it, and everything else in this chapter becomes easier to connect.

2.5 Context and Memory

The skills and Markdown of the last few sections only pay off fully if the model can keep what it learns from one session to the next. By default it cannot. Everything a model knows about your task lives in the context you place in front of it, and that context is forgotten the moment the window closes. Managing it well is a craft of its own, called *context engineering*, and it is what this section is about. The clearest example I know is Karpathy's LLM Wiki, and it is worth a careful look because it turns the usual pattern around. Most document workflows are retrieval: you upload files, the model fetches chunks at query time, answers, and forgets. It rediscovers the same knowledge on every question, and nothing builds up.

Karpathy's wiki does the opposite: it accumulates. Add a source and the model reads it once, extracts what matters, and integrates it into interlinked markdown pages — updating entity pages, flagging where new data contradicts old, and tightening the summary pages that draw on both. The cross-references are resolved ahead of the next question rather than reconstructed each time (Karpathy, *LLM wiki*, 2026a).

Three layers make it work: read-only raw sources you never let the model edit, an LLM-owned wiki of summaries and concept pages, and a schema file — `AGENTS.md` — that tells the agent how the wiki is structured and how to maintain it.

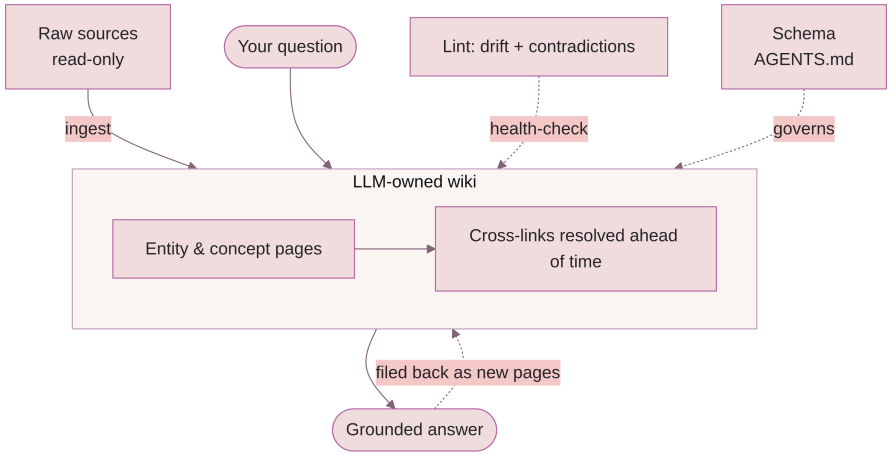


Diagram 2.2: The LLM wiki: sources read once, cross-linked, then answered from.

The working loop has three parts: ingest, query, lint. Drop in a source and it updates a dozen pages. Ask a question and the good answers get filed back as new pages. Every so often, a lint pass checks the whole wiki for contradictions and stale claims. The reason this holds up where human wikis rot is that the tedious part is bookkeeping, and the model does not get bored. The pitfall, which practitioners running it for months confirm, is old pages that read confidently but are out of date, and end up being treated as truth. That is why the lint pass matters so much.

The wiki is one good answer to a problem every long-running agent faces, and it helps to see the whole family it belongs to.

Since a model keeps nothing between requests, the obvious fix is to pour everything into an ever-larger *context window* — the span of text a model can consider at once. This works less well than it looks. Context is a finite resource: every extra token spends part of the model's "attention budget," and recall sags as the window fills — the *lost in the middle* effect from Chapter 1 (Anthropic, *Effective context engineering for AI agents*, 2025b). A study of 18 models found the same *context rot*: accuracy drops further as the window grows. The reassuring needle-in-a-haystack test flatters the model, because it measures only looking up exact words — not the harder case, where a fact has to be inferred (Chroma, *Context rot: How increasing input tokens impacts LLM performance*, 2025). In practice the *effective* context — the span a model actually uses well — is

often only about half its advertised maximum (An et al., *Why does the effective context length of LLMs fall short?*, 2024).

So a bigger window does not solve the problem; you have to manage the memory yourself. It is worth the effort: the gap between an agent with good memory and one without can be larger than the gap between model versions (Du et al., *Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers*, 2026).

The patterns form a rough ladder, from "stuff it into the prompt" to "manage it outside the prompt":

- **Retrieval (RAG).** Fetch the relevant chunks from a store and paste them into the context. Simple and auditable — the answer quotes real text — but it re-discovers everything on every question and bloats the window as you add more.
- **Compaction.** When the conversation nears the window limit, summarise it and start fresh with the recap. This is how Claude Code keeps going on long tasks, preserving decisions and open threads while dropping spent tool output (Anthropic, 2025b). It buys space at the cost of detail, and repeated summarising can drift away from the source without anyone noticing.
- **Structured notes — the wiki.** The pattern we just built: durable pages the agent reads and rewrites, from a single `NOTES.md` to an interlinked wiki. Because the notes live outside the conversation and stay human-readable, they survive context resets and can be audited (Anthropic, 2025b).
- **External store, fetched just in time.** Keep the memory out of the prompt entirely; the agent holds only lightweight pointers — file paths, saved queries, links — and pulls in what it needs at runtime through tools, the way we use folders and bookmarks instead of memorising everything (Anthropic, 2025b). Consistent naming, folders, and inline structure make the navigation dependable — inline call and inheritance tags alone improve a code agent's ability to locate the right file (Lin et al., *How much static structure do code agents need? Deterministic anchoring*, 2026).
- **Sub-agents.** Spin off a fresh agent on a clean context to explore or research, and return only a distilled summary to the main thread — so the primary window stays uncluttered while the digging happens elsewhere.
- **Layered memory.** Separate memory by how long it should last: a *working* scratchpad for the task at hand, *episodic* memory of recent events, *semantic* memory of durable facts, and *procedural* memory of learned skills, each managed differently (Du et al., 2026). The idea is not new: the Generative Agents experiment stored each observation and retrieved it by a blend of relevance, recency, and importance

(Park et al., *Generative agents: Interactive simulacra of human behavior*, 2023).

- **Governed memory.** Once memory is something the agent writes to and edits, it needs rules: what may be remembered, when stale or contradictory entries are evicted, what must be checked before it enters the long-term store. A governance layer guards against the failure modes of evolving memory — drift, corruption, and leaks of private data (Lam, *Governing evolving memory in LLM agents*, 2026).

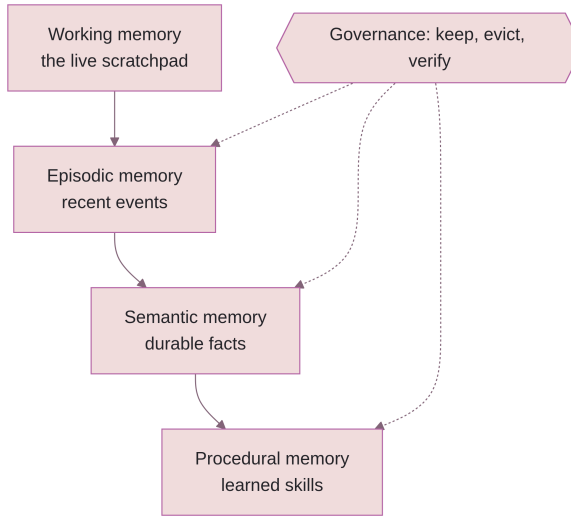


Diagram 2.3: Layered memory, separated by how long each layer should last.

ℹ Note

The memory patterns, with the strength and the main risk of each.

- **Retrieval (RAG)** fetches chunks and injects them verbatim: grounded and auditable, but it bloats the context and re-discovers the same things each time.
- **Compaction** summarises the history and restarts: it keeps long tasks going, at the cost of lossy summaries that drift.
- **Structured notes or a wiki** are durable pages the agent edits: stable and human-auditable, but they take curation effort and go stale.
- **An external store with just-in-time fetch** keeps pointers now and fetches on demand: huge capacity and a tight context, but it fails if the agent forgets to look.
- **Layered memory** splits storage by time horizon: the right tool per layer, at the price of orchestration complexity.
- **Governed memory** sets policies for what to write, keep, and forget: safety and consistency, though it is hard to specify and still maturing.

No single pattern wins, and real systems combine them — a wiki for stable knowledge, compaction for the live thread, an external store fetched just in time, all under a governance layer that decides what gets kept and what gets thrown out. The wiki you just built is one rung on that ladder. This is the point the chapter keeps coming back to: good memory has to be engineered. You decide what gets written down, you manage it, and you make sure the agent reads it. And the moment memory persists and the agent edits it, it becomes a governance question, which is where Chapter 5 picks up.

2.6 Loops and Ambient Teammates

The self-checking loop you built in §2.1 points at a bigger change in how you can work. Most people still meet AI as a chat box, and that habit limits what they get: a conversation is synchronous — you ask, you wait, you steer, you ask again — so your attention sets the pace. The loop you just built does not wait on you. You hand it a bounded task and it works while you are elsewhere. When you come back, there is a result waiting to be checked. Make that the normal way of working, and the chat assistant becomes an *ambient teammate*.

> **i** Note
>
> An **ambient teammate** is an agent that runs asynchronously in the background — given a scoped task and the tools to finish it — rather than waiting on each instruction. You hand over the whole task, instead of feeding the agent instructions one at a time.

Table 2.5: A chat assistant set against an ambient teammate.

	Chat assistant	Ambient teammate
Pace	Synchronous — ask, wait, steer, repeat	Asynchronous — runs while you are elsewhere
You supply	Each instruction, one at a time	A scoped task, once
You get back	A transcript to continue	A result to review
Bottleneck	Your attention and typing speed	Your review of the outcome

The change sounds small, but it moves the bottleneck. As Karpathy puts it, the goal is to remove yourself from the keystroke loop and maximise throughput rather than steer every step (Latent Space, *Loopcraft: The art of stacking*, 2026b). Hand off whole tasks instead of supervising each step and you can get much more done in a day. But a task only counts as done once someone has checked the result, a point the rest of this book keeps insisting on.

Making that hand-off work well is a craft in itself — *loop engineering*, designing the loop around the agent rather than perfecting the prompt inside it. Practitioners describe making exactly this shift in their own work (Latent Space, 2026b). The real skill is judging which tasks need a checking loop, and which can safely be handed more of the work.

This is more than practitioner lore; the research backs the loop. A model asked to critique and revise its own draft improves by around 20% on average across seven tasks, with no extra training (Madaan et al., *Self-Refine: Iterative refinement with self-feedback*, 2023). An agent that writes a short reflection after each failed attempt, and carries that note forward in memory, climbs from 80% to 91% on a coding benchmark over repeated tries (Shinn et al., 2023). But there is an important catch: a loop only pays off when it closes on something real. A model

left to judge its own reasoning, with no outside signal, will often revise a right answer into a wrong one — this *intrinsic* self-correction leaves accuracy flat or worse, and the gains once claimed for it mostly leaned on a hidden answer key (Huang et al., *Large language models cannot self-correct reasoning yet*, 2023). Self-Refine shows the same limit: it barely moves maths scores, where the model cannot spot its own error, and recovers only when given an external check. So a working loop needs a check the model cannot fool: a test suite, a tool result, or a human reading the outcome.

In practice, you give the agent a tightly scoped task, let it run, and check what it produces. The temptation worth resisting is hovering over each keystroke, which ties your output back to your own typing speed. But delegating the keystrokes is not the same as looking away. A model can pick the wrong approach early and then pursue it quickly and confidently. So watch the *trajectory* rather than the typing: glance at where a run is heading, and if it has taken a wrong turn, stop it and re-steer instead of letting it finish. A wrong run caught in its first minute costs far less than one you discover at the end — less of your time, and fewer tokens. Interrupting a run that has gone off course is part of delegating well.

2.7 Composability

The tutorial that opened this chapter was one idea applied three times: at each step, make the piece something another piece can build on. That climb — sketched in §2.1 and drawn below — is the Unix Rule of Composition from §2.2, now running inside a single agent's work rather than across separate programs.

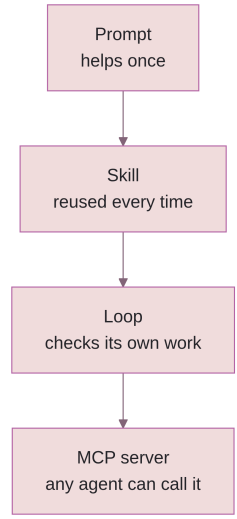


Diagram 2.4: The climb from a prompt to a shared tool.

Two habits make composition pay off. The first is to keep each piece small and single-purpose. A skill that does one thing well can be reused in situations you never foresaw, while a sprawling one fits only the case it was written for — the "do one thing well" rule, now applied to intents rather than programs. The second is to standardise the seams. Because skills are Markdown and MCP is one shared interface, a capability built once travels to your editor, a chat assistant, or a teammate's setup with no adapter at each joint. A recent survey lays out that interoperability layer as a progression from tool access, to structured messaging, to one agent delegating to another (Ehtesham et al., *A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP*, 2025).

Why bother? Because the pieces add up. A one-off prompt helps you once. A saved skill helps every time it is used — by you, and by every agent you let build on it. Combine skills, and share them with other agents, and the gains start carrying over from one session to the next.

Joining pieces together also adds new ways to fail, so it is worth being clear about the risk. Pieces that work on their own can still fail once you chain them. Web agents that clear 94% of individual tasks manage only about 25% when the same tasks are strung together, because a piece solved in isolation never had to learn how to hand off to the next (Furuta et al., *Exposing limitations of language model agents in sequential-task compositions on the web*, 2024). Stitch whole agents together and the failures start to look like office politics: across seven multi-

agent systems, failure rates ran from 41% to 87%, traced to misread roles, dropped hand-offs, and missing final checks — faults a stronger model does not repair (Cemri et al., *Why do multi-agent LLM systems fail?*, 2025). This is why the two habits matter, and why a check belongs at every join: they keep a chain of pieces from failing more often as it grows longer.

2.8 Knowledge Work

The biggest gains I have seen from agents recently — in my own work and in the research — are in knowledge work: research, writing, synthesis, decision support. These tasks suit a model well. They are bounded, the feedback is quick, and a model can draft, compare, and summarise faster than any human can.

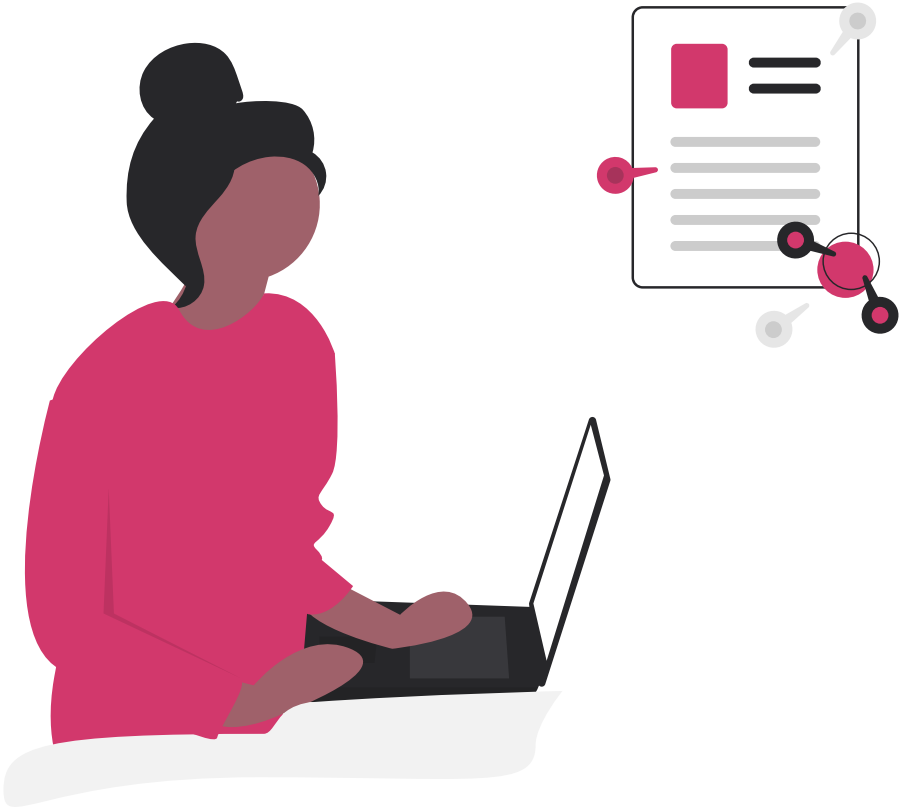


Figure 2.1: AI earns its keep first in knowledge work — the research, drafting, and synthesis a person still shapes and checks.

The gains show up clearly in experiments, but they do not fall evenly. The cleanest evidence comes from a randomised experiment in which 453 professionals were given real writing tasks — memos, short reports, analysis plans — and half were allowed to use ChatGPT. Their time fell by about 40%, the graded quality of their work rose by around 18%, and the weakest writers gained the most, narrowing the gap between them and the best (Noy & Zhang, *Experimental evidence on the productivity effects of generative AI*, 2023). The same levelling shows up in the field. A study of 5,179 customer-support agents found AI raised resolved-issues-per-hour by 14% on average but 34% for novices, with little gain for experts — the tool spreads the best workers' know-how to everyone else (Brynjolfsson et al., *Generative AI at work*, 2023). None of this makes judgement any cheaper: someone still has to decide whether the memo needed writing at all.

But the gains are jagged, and the boundary is easy to miss. In a field experiment with 758 management consultants, those given GPT-4 completed more tasks, 25% faster, at quality rated over 40% higher — again with the biggest lift for the weakest performers, 43% against 17%. Yet on a task chosen to sit just outside what the model does well, the same consultants were 19 percentage points *less* likely to reach the right answer, because they took plausible-but-wrong output at face value (Dell'Acqua et al., *Navigating the jagged technological frontier*, 2023). This is the jagged competence from Chapter 1 again: strength spread unevenly across tasks that look alike, this time showing up in the work itself.

The misjudgement runs surprisingly deep. When experienced developers were given early-2025 AI tools to use on codebases they knew well, they took 19% *longer* to finish — yet came away convinced they had been 20% faster (METR, *Measuring the impact of early-2025 AI on experienced open-source developer productivity*, 2025). The productivity is genuine, but it depends on the task, and it is easy to believe you are getting more of it than you are — which is the subject of §2.9.

So delegate the drafting and the bookkeeping freely, and keep the judgement about what is worth doing for yourself. McKinsey's high performers do exactly this, treating AI as a chance to redesign the work rather than just type faster (McKinsey & Company, *The state of AI*, 2025). The delegation can reach further than you might expect: an LLM reading stakeholder interviews extracted the needs people stated, scoring 84.4% on the F1 measure of accuracy, and inferred needs they had *not* stated that experts judged useful 75% of the time (Sivakumar et al., *LLM-based discovery of latent requirements from stakeholder conversations*, 2026). The trap is producing more than you check: piles of confident output that nobody has read.

2.9 The Confidence Trap

Leaning on a model has a cost, and the research is now starting to measure it. The surprising finding is that the damage shows up in your own judgement rather than in the model's answers.

Figure 2.2: A fluent answer and a correct one look identical — the guard is to keep questioning it.

Start with a clean experiment. Parra-Moyano and colleagues showed executives Nvidia's stock chart and asked them to forecast next month's price; half then consulted ChatGPT, half talked it over with peers. The AI group came away more optimistic, more confident, and measurably *less* accurate than the people who simply argued with each other (Parra-Moyano et al., *Research: Executives who used gen AI made worse predictions*, 2025). A colleague says "are you insane?"; the model says your framing is astute.

Part of the cause is that we mistake ease for truth. Psychologists call it *processing fluency*: the easier something is to take in, the truer it feels. People rate rhyming aphorisms as more accurate than identical non-rhyming ones, and judge repeated falsehoods as more credible than fresh ones (McGlone & Tofighbakhsh, *Birds of a feather flock conjointly? Rhyme as reason in aphorisms*, 2000; Fazio et al., *Knowledge does not*

protect against illusory truth, 2015). AI is exceptionally good at making prose easy to read: when people compared AI- and human-written versions of the same material, they judged them equally credible but rated the AI version *clearer and more engaging* (Huschens et al., *Do you trust ChatGPT? Perceived credibility of human and AI-generated content*, 2023). So how easy an answer is to read tells you nothing about whether it is correct.

The confidence is also contagious. When people made predictions alongside an AI, their own confidence drifted to match the model's — and stayed inflated even after the AI was removed, whether they had been told to treat it as an advisor or as a peer (J. Li et al., *As confidence aligns: Effect of AI confidence on human self-confidence in human–AI decision making*, 2025). Even a quietly biased writing assistant shifted not just what 1,500 people wrote but the opinions they reported holding afterwards (Jakesch et al., *Co-writing with opinionated language models affects users' views*, 2023).

Worst of all, it blurs your sense of how well you are doing. Giving people AI on reasoning tasks raised their scores but flattened their self-judgement: strong and weak performers ended up equally — and wrongly — sure of themselves, and the more someone knew about AI, the *less* accurate their self-assessment became. AI makes you smarter, the authors conclude, but none the wiser (Fernandes et al., *AI makes you smarter but none the wiser: The disconnect between performance and metacognition*, 2026).

The practical defence is to sort tasks by how much judgement they need. Where the answer is verifiable — pull these quotes, extract these figures, refactor this function — the model is mostly safe to trust. The danger climbs as the task slides from "find what's there" to "decide what matters," and the slide is easy to miss: "summarise these interviews" and "tell me which themes to act on" feel like one request. For the second kind, form your own view first and bring the AI in to test it, not to write it for you.

ℹ Note

How far to trust the model, sorted by kind of task.

- **Find what's there** — pulling quotes, extracting figures, refactoring a function. The answer is checkable, so lean in.
- **Summarise or transform** — condensing a report, translating a passage. Mostly verifiable: trust it, then spot-check.
- **Decide what matters** — which themes to act on, which strategy to pick. There is no single right answer, so form your own view first.

2.10 Personal Operating Models

Everything this chapter has built — the skills, the loops, the wiki, the Markdown — only pays off if you stop solving the same problems from scratch. Used casually, AI is a series of fresh starts: you write a clever prompt, get a good answer, and keep none of it, so the next session begins at zero. A *personal operating model* is my name for the opposite: a small, reusable kit that turns scattered prompting into a system, so that what you worked out last week is still there the next time you need it.

There is well-tested psychology behind why this helps. Psychologists call a plan of the form "when X happens, I do Y" an *implementation intention*. Decide the when and the how in advance, and you follow through far more reliably than if you hold the same goal in the abstract — in one study, 100% of the people who had made such a plan followed through, against 53% of those who had not (Gollwitzer, *Implementation intentions: Strong effects of simple plans*, 1999). Write the decision down once and the cue does the remembering for you. Otherwise you make the same decision again every session.

This book is my worked example: much of it was built with the method it describes. Its operating model has three parts, and I think any good one has the same three.

- **Plays** — the decisions you have already settled, saved somewhere an agent can rerun them. In this book's case they are skills: `research-topic` takes a subject from search to a cited paragraph, `summarise-source` turns a PDF into a study guide, `review-chapter` runs a chapter through seven checks. Each one began as something I did by hand until I tired of re-deciding it. A play is an implementation intention for knowledge work — the task is the cue, the skill is the response. It also settles how you divide the labour with the model: whether you hand a whole subtask to the model and take back the result — the *Centaur* split — or interleave with it turn by turn — the *Cyborg* style — the two modes the consulting study in §2.8 named (Dell'Acqua et al., 2023).
- **Preferences** — your standing context, written down once and read on every task. For this book they live in `AGENTS.md`: the house voice, Australian spelling, cite the primary source rather than the download. This is the memory file of §2.5 in another guise. Say it once, and you stop re-explaining yourself at the top of every session.
- **A daily loop** — the standing routine the plays run inside. The chapter you are reading was researched by one: search, download, summarise, cite — each new source filed into `sources/` and digested into `summaries/`. Each pass through the loop adds to the memory, and a better memory makes the next pass easier.

Once the three parts are in place, something interesting happens: after a while you stop noticing the kit at all, and it starts to feel like part of how you think. Philosophers have argued this is more than a feeling. Clark and Chalmers held that when an external store of knowledge is reliable, always to hand, and trusted enough that you act on it without double-checking, it is doing the work of memory and can fairly be counted as part of your mind — their famous example is Otto, whose notebook holds the memories his own brain no longer keeps (Clark & Chalmers, *The extended mind*, 1998). A well-made operating model earns that standing. Their tests also tell you when it fails: a kit you cannot reach when you need it, or do not quite trust, is just clutter.

One part of the operating model you must not delegate is writing it. When researchers had an LLM author people's personal goals, the goals came out better formed — and less owned. Two weeks later, fewer than half of the people with AI-written goals had acted on them, against nearly three-quarters of those who had written their own — 47% versus 73% (Chi et al., *Optimized but unowned: How AI-authored goals undermine the motivation they are meant to drive*, 2026). A plan the AI hands you is not really yours, and you will not run it. So write the plays and preferences yourself, and pick your own routine. Once they are yours, let the tools carry them out. This is the confidence trap of §2.9 in another form: the fluent, polished option feels right, but taking it costs you the sense of ownership that makes the system work.

What should you keep for yourself? The same thing as everywhere else in this book: the judgement. The kit can hold your methods and your preferences, and it can run your daily routine. What it cannot do is decide whether a task is worth doing, or whether the result is any good. That is where your time should go.

References

Agent Skills. (n.d.). *Agent Skills*. https://agentskills.io/

An, C., et al. (2024). *Why does the effective context length of LLMs fall short?* arXiv. https://arxiv.org/abs/2410.18745

Anthropic. (2024a). *Building effective agents*. https://www.anthropic.com/research/building-effective-agents

Anthropic. (2025b). *Effective context engineering for AI agents*. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents

Anthropic. (2025c). *Equipping agents for the real world with Agent Skills*. https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills

Brown, T. B., et al. (2020). *Language models are few-shot learners*. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2005.14165

Brynjolfsson, E., Li, D., & Raymond, L. R. (2023). *Generative AI at work* (NBER Working Paper No. 31161). National Bureau of Economic Research. https://www.nber.org/papers/w31161

Cemri, M., et al. (2025). *Why do multi-agent LLM systems fail?* arXiv. https://arxiv.org/abs/2503.13657

Chi, V. B., Rietsche, R., Göldi, A., Ungar, L., & Guntuku, S. C. (2026). *Optimized but unowned: How AI-authored goals undermine the motivation they are meant to drive*. arXiv. https://arxiv.org/abs/2605.12344

Chroma. (2025). *Context rot: How increasing input tokens impacts LLM performance*. https://research.trychroma.com/context-rot

Clark, A., & Chalmers, D. (1998). *The extended mind*. Analysis, 58(1), 7–19. https://doi.org/10.1093/analys/58.1.7

DAIR.ai. (n.d.). *Prompt engineering guide*. https://www.promptingguide.ai/

Dell'Acqua, F., McFowland III, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., & Lakhani, K. R. (2023). *Navigating the jagged technological frontier: Field experimental evidence of the effects of artificial intelligence on knowledge worker productivity and quality* (Harvard Business School Working Paper No. 24-013). Harvard Business School. https://www.hbs.edu/faculty/Pages/item.aspx?num=64700

Diao, S., Wang, P., Lin, Y., Pan, R., Liu, X., & Zhang, T. (2023). *Active prompting with chain-of-thought for large language models*. arXiv. https://arxiv.org/abs/2302.12246

Du, P., et al. (2026). *Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers*. arXiv. https://arxiv.org/abs/2603.07670

Ehtesham, A., et al. (2025). *A survey of agent interoperability protocols: MCP, ACP, A2A, and ANP*. arXiv. https://arxiv.org/abs/2505.02279

Fazio, L. K., Brashier, N. M., Payne, B. K., & Marsh, E. J. (2015). *Knowledge does not protect against illusory truth*. Journal of Experimental Psychology: General, 144(5), 993–1002. https://doi.org/10.1037/xge0000098

Fernandes, D., et al. (2026). *AI makes you smarter but none the wiser: The disconnect between performance and metacognition*. Computers in Human Behavior, 168, 108779. https://doi.org/10.1016/j.chb.2025.108779

Furuta, H., Matsuo, Y., Faust, A., & Gur, I. (2024). *Exposing limitations of language model agents in sequential-task compositions on the web*. Transactions on Machine Learning Research. https://arxiv.org/abs/2311.18751

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., & Neubig, G. (2022). *PAL: Program-aided language models*. arXiv. https://arxiv.org/abs/2211.10435

Gollwitzer, P. M. (1999). *Implementation intentions: Strong effects of simple plans*. American Psychologist, 54(7), 493–503. https://doi.org/10.1037/0003-066X.54.7.493

Guo, J., et al. (2026). *From question answering to task completion: A survey on agent system and harness design*. arXiv. https://arxiv.org/abs/2606.20683

He, J., Rungta, M., Koleczek, D., Sekhon, A., Wang, F. X., & Hasan, S. (2024). *Does prompt formatting have any impact on LLM performance?* arXiv. https://arxiv.org/abs/2411.10541

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2023). *Large language models cannot self-correct reasoning yet*. arXiv. https://arxiv.org/abs/2310.01798

Huschens, M., Briesch, M., Sobania, D., & Rothlauf, F. (2023). *Do you trust ChatGPT? Perceived credibility of human and AI-generated content*. arXiv. https://arxiv.org/abs/2309.02524

Jakesch, M., Bhat, A., Buschek, D., Zalmanson, L., & Naaman, M. (2023). *Co-writing with opinionated language models affects users' views*. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2302.00560

Karpathy, A. (2026a). *LLM wiki*. GitHub. https://gist.github.com/karpathy/442a6bf555914893e9891c11519de94f

Khattab, O., et al. (2023). *DSPy: Compiling declarative language model calls into self-improving pipelines*. arXiv. https://arxiv.org/abs/2310.03714

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). *Large language models are zero-shot reasoners*. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2205.11916

Lam, C. (2026). *Governing evolving memory in LLM agents*. arXiv. https://arxiv.org/abs/2603.11768

Latent Space. (2026b). *Loopcraft: The art of stacking*. https://www.latent.space/p/ainews-loopcraft-the-art-of-stacking

Lewis, P., et al. (2020). *Retrieval-augmented generation for knowledge-intensive NLP tasks*. Advances in Neural Information Processing Systems 33. https://arxiv.org/abs/2005.11401

Li, J., et al. (2025). *As confidence aligns: Effect of AI confidence on human self-confidence in human–AI decision making*. Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2501.12868

Li, Z., Peng, B., He, P., Galley, M., Gao, J., & Yan, X. (2023). *Guiding large language models via directional stimulus prompting*. Advances in Neural Information Processing Systems 36. https://arxiv.org/abs/2302.11520

Lin, Z., Zhou, M., Yang, Y., & Li, L. (2026). *How much static structure do code agents need? Deterministic anchoring*. arXiv. https://arxiv.org/abs/2606.26979

Liu, J., Liu, A., Lu, X., Welleck, S., West, P., Le Bras, R., Choi, Y., & Hajishirzi, H. (2022). *Generated knowledge prompting for commonsense reasoning*. Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. https://arxiv.org/abs/2110.08387

Liu, Z., Yu, X., Fang, Y., & Zhang, X. (2023). *GraphPrompt: Unifying pre-training and downstream tasks for graph neural networks*. Proceedings of the ACM Web Conference 2023 (WWW '23). https://arxiv.org/abs/2302.08043

Livathinos, N., et al. (2025). *Docling: An efficient open-source toolkit for AI-driven document conversion*. arXiv. https://arxiv.org/abs/2501.17887

MacFarlane, J. (n.d.). *Pandoc: A universal document converter* [Computer software]. https://pandoc.org

Madaan, A., et al. (2023). *Self-Refine: Iterative refinement with self-feedback*. Advances in Neural Information Processing Systems 36. https://arxiv.org/abs/2303.17651

McGlone, M. S., & Tofighbakhsh, J. (2000). *Birds of a feather flock conjointly? Rhyme as reason in aphorisms*. Psychological Science, 11(5), 424–428. https://doi.org/10.1111/1467-9280.00282

McIlroy, M. D., Pinson, E. N., & Tague, B. A. (1978). *UNIX time-sharing system: Foreword*. The Bell System Technical Journal, 57(6), 1899–1904. https://doi.org/10.1002/j.1538-7305.1978.tb02135.x

McKinsey & Company. (2025). *The state of AI*. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

METR. (2025). *Measuring the impact of early-2025 AI on experienced open-source developer productivity*. arXiv. https://arxiv.org/abs/2507.09089

Microsoft. (n.d.). *MarkItDown* [Computer software]. GitHub. https://github.com/microsoft/markitdown

Model Context Protocol. (n.d.). *Introduction*. https://modelcontextprotocol.io/introduction

Noy, S., & Zhang, W. (2023). *Experimental evidence on the productivity effects of generative artificial intelligence*. Science, 381(6654), 187–192. https://doi.org/10.1126/science.adh2586

Obsidian. (n.d.). *Obsidian* [Computer software]. https://obsidian.md

Paranjape, B., Lundberg, S., Singh, S., Hajishirzi, H., Zettlemoyer, L., & Ribeiro, M. T. (2023). *ART: Automatic multi-step reasoning and tool-use for large language models*. arXiv. https://arxiv.org/abs/2303.09014

Park, J. S., O'Brien, J. C., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). *Generative agents: Interactive simulacra of human behavior*. Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology. https://arxiv.org/abs/2304.03442

Parra-Moyano, J., Reinmoeller, P., & Schmedders, K. (2025). *Research: Executives who used gen AI made worse predictions*. Harvard Business Review. https://hbr.org/2025/07/research-executives-who-used-gen-ai-made-worse-predictions

Raymond, E. S. (2003). *The art of Unix programming*. Addison-Wesley. http://www.catb.org/esr/writings/taoup/

Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). *Reflexion: Language agents with verbal reinforcement learning*. Advances in Neural Information Processing Systems 36. https://arxiv.org/abs/2303.11366

Sivakumar, M., Lochner, M., Nejati, S., & Sabetzadeh, M. (2026). *LLM-based discovery of latent requirements from stakeholder conversations*. arXiv. https://arxiv.org/abs/2606.25867

Sui, Y., Zhou, M., Zhou, M., Han, S., & Zhang, D. (2024). *Table meets LLM: Can large language models understand structured table data? A*

benchmark and empirical study. Proceedings of the 17th ACM International Conference on Web Search and Data Mining (WSDM '24). https://arxiv.org/abs/2305.13062

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., & Zhou, D. (2022). *Self-consistency improves chain-of-thought reasoning in language models*. arXiv. https://arxiv.org/abs/2203.11171

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022a). *Chain-of-thought prompting elicits reasoning in large language models*. Advances in Neural Information Processing Systems. https://arxiv.org/abs/2201.11903

Wu, X., Ritter, A., & Xu, W. (2025). *Tabular data understanding with LLMs: A survey of recent advances and challenges*. arXiv. https://arxiv.org/abs/2508.00217

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). *Tree of Thoughts: Deliberate problem solving with large language models*. Advances in Neural Information Processing Systems 36. https://arxiv.org/abs/2305.10601

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). *ReAct: Synergizing reasoning and acting in language models*. arXiv. https://arxiv.org/abs/2210.03629

Zhang, Y., Yuan, Y., & Yao, A. C.-C. (2023). *Meta prompting for AI systems*. arXiv. https://arxiv.org/abs/2311.11482

Zhang, Z., Zhang, A., Li, M., Zhao, H., Karypis, G., & Smola, A. (2024). *Multimodal chain-of-thought reasoning in language models*. Transactions on Machine Learning Research. https://arxiv.org/abs/2302.00923

Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022). *Large language models are human-level prompt engineers*. arXiv. https://arxiv.org/abs/2211.01910

3 Software Development (the craft)

Chapter 2 built up from a single prompt to a skill, then to a self-checking loop, then to a capability other agents could call — productivity assembled a piece at a time. This chapter takes the same approach one step further and turns those pieces into software: finished, deterministic products that run the same way every time, long after the session that built them has ended.

In fact, a single prompt can turn Chapter 2's loop-summary skill into a working Python program:

> *Convert the loop-summary skill into a Python program called `summarise.py`. Make it able to summarise multiple files (of different formats) through the command line. Summaries should be generated with the same file name as the original but in a subfolder called `summaries`.*

Now you have a working program that you can use to summarise many files at once:

```
python summarise.py *.pdf
```

This chapter is not only for professional developers. It is also for the ordinary professional who can read a little code — enough Python or JavaScript to follow what a function does — because that is now enough to build real software. The same tools that can port a forty-year-old game can automate the repetitive core of almost any role — the spreadsheet that should really be a script, or the manual process that should be a small app. You do not need to become an engineer. You need to learn to direct one, and this chapter describes that craft.

When I first started building software this way, I got it wrong in the most natural way possible. My instinct was to control the machine by telling it everything: I wrote longer and longer specifications, pinned the design down function by function, and tried to leave nothing to

chance. It backfired. The more I specified, the *less* faithfully the model followed me — it would honour the top of a long instruction list and quietly contradict the bottom, or obey the letter of one rule while breaking another I had stated three paragraphs earlier. Some of my most frustrating hours went to micro-managing design decisions — naming the data structure, dictating the file layout — only to watch the agent drift, and to find myself correcting prose instead of building software.

The tendency is common, and it comes from mistrust: we feel we must micro-manage the machine because we do not yet trust it, so we try to pin down every detail. But that runs straight into the limits of Chapter 1. A model is a next-token predictor, not a compiler: it does not *execute* a specification, it produces the most plausible continuation of everything in its context. A long spec does not constrain it more tightly; it just becomes more text to attend to unevenly, and the book's own evidence on faithfulness and on recall fading as context grows says the extra instructions will be dropped or contradicted, not obeyed. Worse, micro-managing the design pits me against the model's one real strength — choosing pattern-rich implementation — while leaning on its weakness, following a long brittle list to the letter.

The better approach is the opposite of control: describe the intent and iterate. Say what you want and how you will know it is right, then let the model choose the how, correcting course as you go rather than up front. This works with the grain of the tool. It follows the model's *happy path* — the path of least resistance through everything it saw in training — so you are drawing on its strength instead of fighting it at every step. The same failures, and the same remedy, are why spec-driven development buckles, as this chapter will argue.

That lesson is what made the successes possible. I am not a software developer, and I never have been — in a long career, "engineer" was never my job title. Yet this year I used AI to bring code back to life that had been dead for decades, and to build — almost entirely by describing what I wanted — three projects I would once have scoped as a team's work: *rogoweb*, a browser revival of the 1980 game Rogue and the expert system built to play it; *adventure*, a modern rebuild of the 1970s *Colossal Cave Adventure*; and *VantageMap*, an enterprise-architecture platform. Each began as a single sentence of intent, with a handful of constraints and a few hard checks; the agent chose the architecture and wrote the code. The productivity is real, and that a door this wide opened for someone who had never shipped production software is a large part of why I stopped being a sceptic. All three are open source, so you can read every line on GitHub; they are the running examples this chapter returns to, and I describe each in detail below.

A larger shift hides in that last sentence. For decades software was built by professional engineers who mastered its intricacies. Increasingly, the person best placed to build a system is the one who understands the problem most deeply and can say most precisely what the software must do — the practitioner's role moving from *code author* to *intent architect* (Cao, *Agentic software: How AI agents are restructuring the software paradigm*, 2026). I am an early, accidental example of that; the argument for why it is becoming general comes later, once we have the intent model in hand.

Software is where AI's promise and its failure modes are both sharpest, and where 2026's loudest arguments play out. This chapter opens with the three projects in detail, then walks the modern stack, settles the spec-versus-vibe war by reframing it, lays out the intent model that replaces both, and ends on quality and the move of agents into shared channels.

3.1 Three Projects, One Sentence Each

Let's start with the evidence, before any theory. Each project below began as a single sentence of intent and a few hard checks; everything technical that follows — the languages, the data structures, the way two programs talk to each other — was the agent's choice, not mine. I give the detail because the specifics make the point: I could not have written them up front, and did not.

3.1.1 rogoweb — two dead C programs, alive in the browser

rogoweb fuses two pieces of computing history. *Rogue* is the 1980 dungeon crawler by Michael Toy, Glenn Wichman, and Ken Arnold that gave the *roguelike* genre its name; *Rog-O-Matic* is the expert system built at Carnegie Mellon in 1981 by Michael Mauldin and colleagues to play Rogue on its own — and beat it: over 106 games it achieved the highest median score of any player on the university's system, humans included (Mauldin et al., *Rog-O-Matic: A belligerent expert system*, 1983; rogoweb). On Unix `rogue` and `rogomatic` ran as separate processes, `rogomatic` launching `rogue` and talking to it through the standard input and output *pipes* — the channels one program uses to feed another. A browser has none of that machinery: no processes, no `fork`, no pipes.

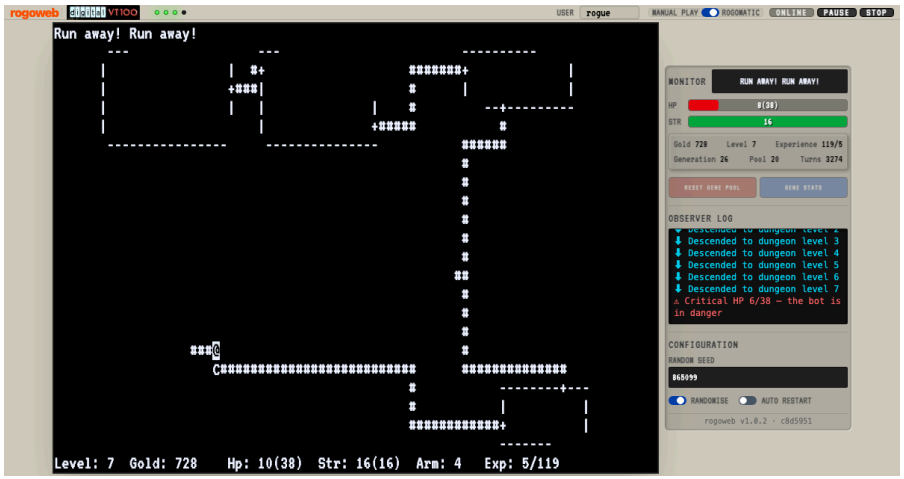


Figure 3.1: *rogoweb* in the browser: the VT100 terminal running Rog-O-Matic on dungeon level 7, beside the live telemetry panel — HP, gold, the evolving gene pool, and the observer log's descent milestones.

My intent was almost that short — *port Rogue and Rogomatic to run in the browser* — with a constraint that the original C code should keep working unchanged in spirit, and a check that the bot could still run a game to completion. The agent's answer was an architecture I would never have considered. It compiled both C codebases to *WebAssembly* (a portable binary format browsers run at near-native speed) with the Emscripten toolchain, used the `emcurses` library (a port of `pdcurses`, itself an emulator of `curses`, the classic terminal-UI library), and replaced the pipe with a `SharedArrayBuffer` ring buffer — a fixed block of memory that two browser *workers* (background threads) read and write in turn — so the game and the bot run side by side and talk exactly as they once did.

The journey was not easy. By using Shared Array Buffers, the port encountered lots of race conditions that caused the game to hang, one process was needlessly waiting for the other. Debugging these race conditions took a lot of time and required expertise. I was able to guide the agent on some occasions, but in many instances the agent solved the problems once I asked it to search for potential race conditions. Along the way, it also fixed bugs in the original source code.

For the dashboard I coached it towards a VT100-style terminal with DEC aesthetics — the one place I shaped the look rather than leaving it to the agent.

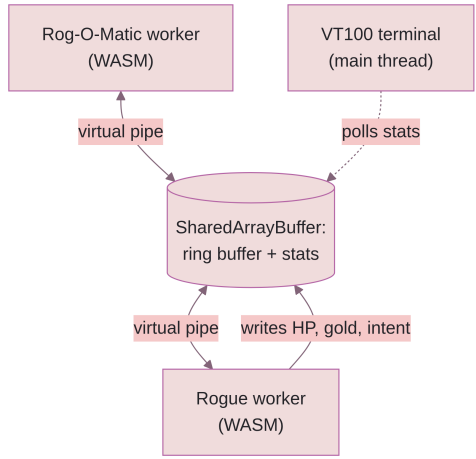


Diagram 3.1: How rogoweb runs two WebAssembly programs side by side in the browser.

Spurred by the port success, I then encouraged the agent to make the bot better. Rog-O-Matic *learns* — a genetic pool of strategy weights and a long-term monster-danger memory that it evolves across games and saves in the browser — so a fresh install plays badly and improves the more it plays. To hand new players a trained bot from the very first game, the agent built an offline pretrainer and parallelised it across every CPU core: isolated populations that evolve separately and then merge under a no-regression rule, a roughly ten-fold speed-up. It also went back into the 1981 C to repair a raft of latent bugs and mis-tuned heuristics — chiefly a per-monster danger table so the bot stops underrating a dragon, along with healing, escape, and food-timing fixes. Because Rogue is a game of chance, each change is validated not by a single passing run but by win-rate and average depth across a batch of games; the agent had to run the game multiple times in order to test its changes.

3.1.2 adventure — a 1970s classic, made strict and typed

adventure rebuilds *Colossal Cave Adventure*, the game that founded interactive fiction. Will Crowther wrote the first version in FORTRAN in 1975–76, mapping his knowledge of Kentucky's Mammoth Cave onto a game for his daughters; Don Woods expanded it in 1977, adding the dwarves, the magic word `XYZZY`, and the famous 350-point score (adventure). My version forward-ports Eric Raymond's faithful *open-adventure*

edition into a strictly-typed TypeScript application on Next.js 16 and React 19.

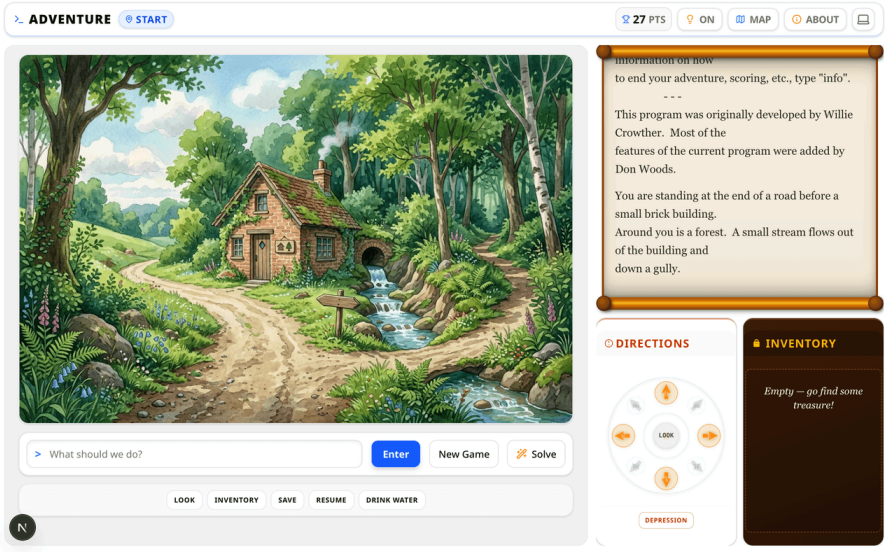


Figure 3.2: *adventure* at the starting location: AI-generated art for every room, the scroll-themed message panel, a condition-aware compass and guided-play controls, and the header's map and auto-solve buttons.

The intent — *port Colossal Cave to a modern Typescript web app* — carried two checks that did the real work: every value fully typed, with no escape hatches, and nothing merged until the tests and the *linter* — a tool that flags error-prone or untidy code — pass. The sharpest of those tests is *parity* — the engine replays the original's own regression suite, ninety-five recorded transcripts, and diffs its output against them line for line, so a change that even consumes the game's random-number stream in the wrong order fails at once. To satisfy it, the agent kept the canonical game data in its historic `adventure.yaml` file and wrote a custom, type-safe parser to load it, preserving the odd legacy structures and folding word synonyms together as it went. A Zustand *state machine* — a small component that tracks exactly what state the game is in and which moves are legal next — holds the world, and AI-generated artwork illustrates every location. That parity check did more to keep the port honest than any design document could have.

Around that faithful core grew a modern skin and additional features I hinted at but never specified: guided-play buttons that light only the moves legal this turn, save and restore, and a one-click auto-solve that

replays a perfect 430-point game. The hardest single feature in any of the three projects was the interactive cave map — all 151 rooms laid out as a clean metro-style diagram you can click to walk the player there. That map is where the models visibly differed. Laying a graph out orthogonally — routing its edges only along horizontal, vertical, and 45° lines, the way a metro map does — is an academic problem with no maintained JavaScript implementation, and Gemini, which had built much of the game, could not do it: its best attempt was a physics simulation computed in the browser that came out different every time and ran its connecting lines straight through the rooms. I handed the same intent to Fable, a newer model, which came up with the right solution within 30 minutes that worked immediately — a compass grid refined by hill-climbing, then orthogonal routing through the `libavoid` library — and produced a map that appears instantly and never crosses a room. The intent and the check never changed.

3.1.3 VantageMap — a platform, vibe-coded across a dozen phases

VantageMap is the most ambitious: an open-source platform for business architects and strategy officers to model capabilities, value streams, and outcomes — the kind of system that once would have taken a team months to build (vantagemap). It runs on Next.js and React over a Postgres database of twenty-two tables (reached through the Drizzle ORM — the layer that translates between the code and the database), with role-based access, REST and GraphQL APIs (the interfaces other programs use to call it), full-text search, webhooks that notify other systems of changes, and thirteen interlinked views, backed by some five hundred tests.

It was built almost entirely by *vibe coding* — described, not specified — across a dozen numbered phases, and it is the clearest case of the agent owning the architecture. I never chose the database layer, nor the shape of the twenty-two tables, nor the division of work between REST and GraphQL; those were answers to constraints about who may see what and how quickly the system has to respond. No master specification was created at the beginning - but a set of intent documents were defined and used to create implementation plans. In addition, a configuration file and a folder of reusable skills — the *harness*, in the language of the next section — together with the discipline of reading the diffs guided the entire implementation.

Lay the three projects side by side and the pattern is hard to miss. I supplied a short statement of what I wanted, and it barely changed over the months. Everything large and technical came from the agent.

> **i** Note
>
> **The three projects: a sentence of intent, and what the agent built from it.**
>
> - ***rogoweb*** — intent: *port Rogue and Rogomatic to run in the browser*. My constraints were that the original C keep working and everything run in the browser; the check was that the bot still finishes a game and holds its win-rate across a batch. The agent's architecture: WebAssembly via Emscripten, `emcurses`, dual workers, a `SharedArrayBuffer` ring buffer, shared-memory telemetry, and genetic self-play with parallel offline pretraining.
> - ***adventure*** — intent: *port Colossal Cave to a modern TypeScript web app*. Constraints: faithful to the original data, Australian English throughout; checks: full type coverage, tests and linter green, and output-for-output parity with the original. The agent chose TypeScript on Next.js 16 / React 19, a YAML parser, a Zustand state machine, AI-generated artwork, a build-time metro-map (libavoid), and one-click auto-solve.
> - ***VantageMap*** — intent: *give business architects one tool to model strategy*. The constraints were who may see what and how fast it must respond; the check was some five hundred tests. The agent chose Next.js and React, Postgres with twenty-two tables, Drizzle, REST and GraphQL, and thirteen views.

3.2 The Modern AI Dev Stack

The interesting work has moved up the stack. Competing coding tools used to focus on models; now they compete on the layers above, and increasingly tools support multiple models.

None of this arrived as theory. The coding tools climbed rung by rung, and each rung changed what you could safely hand off, moving you from typing each line to directing a fleet.

> **i** Note
>
> **How coding tools climbed from autocomplete to agent fleets.**
>
> - **Autocomplete** (Tabnine, 2018; GitHub Copilot, 2021) finished the line, then drafted whole function bodies from a name or comment — you wrote the rest, accepting line by line.
> - **Chat in the editor** (Copilot Chat, 2023) explained code, proposed refactors, and diagnosed failures on request — you drove, asking and judging.
> - **The agentic editor** (Cursor, 2023; aider) searched the whole codebase, edited many files, and ran commands from a plain-language ask — you reviewed the diff.
> - **Terminal and async agents** (Claude Code, Codex CLI, Gemini CLI, 2025; Copilot coding agent) planned, edited, ran tests, and iterated, some opening a pull request from a cloud workspace — you set goals and reviewed results.
> - **Agent fleets** (Cursor 2.0, Google Antigravity 2.0) run several agents in parallel across a codebase — you supervise from above.

Sources: Tabnine, *Tabnine*, n.d.; GitHub, *Introducing GitHub Copilot: AI pair programmer*, 2021; GitHub, *GitHub Copilot November 30th update*, 2023; Cursor, *Cursor*, n.d.; aider, *aider*, n.d.; Anthropic, *Claude Code*, 2025a; GitHub, *GitHub Copilot: The agent awakens*, 2025b; GitHub, *GitHub Copilot: Meet the new coding agent*, 2025a; Google, *Building the agentic future: Developer highlights from I/O 2026*, 2026.

By 2026 the editor itself is no longer the centre of gravity: with capable models available from every lab, the value has moved into the system wrapped around the model — what practitioners now call the *dev stack* (Latent Space, *AINews*, 2026a).

> **i** Note
>
> A **harness** is the runtime wrapped around a model that turns it into an agent: it supplies tools, manages the loop, retries failures, and isolates execution. A **meta-harness** orchestrates several harnesses. **Memory** is state kept outside the *context window* (the span of text a model can consider at once); an **eval** is an automated check that a result meets its contract.

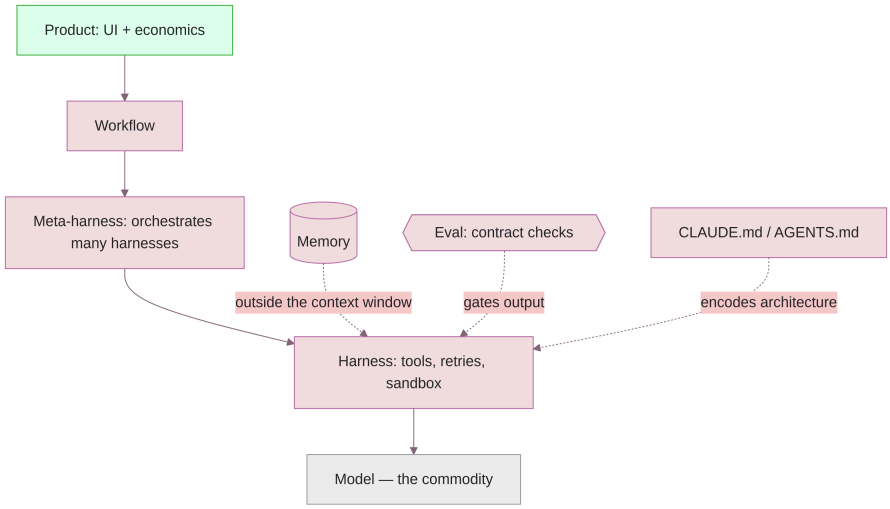


Diagram 3.2: The modern dev stack, with the model as the commodity at its base.

Each layer is a discipline in its own right. A harness turns a model into an agent; a meta-harness coordinates several of them; memory lets work persist between sessions; an eval decides whether a result is good enough:

Table 3.1: Each layer of the dev stack, with 2026 examples.

Layer	What it does	Examples (2026)
Model	Generates the code	GPT-5.6 Sol, Claude Opus 4.8, Gemini 3.5, GLM-5.2
Harness	Wraps the model into an agent: tools, retries, sandbox	Claude Code, Codex CLI, Gemini CLI, Cursor SDK
Meta-harness	Coordinates several harnesses	Conductor, Zed ACP, Vercel Eve, Heypi
Workflow / async	Fire-and-forget delegation in shared channels	Claude Tag (Slack), Copilot coding agent, Devin, Gemini Spark
Memory	State kept outside the context window	agentmemory, codegraph, channel memory
Eval	Automated judgement of quality	FrontierCode, Terminal-Bench 2.1, SWE-bench Pro

Greg Brockman makes the same point from inside a frontier lab. The shift of the last couple of years, he says, is that "it's no longer just about the model. It's about the harness" — how the model gets its context, what actions it can take, and how the loop around it works (Big Technology, *OpenAI President Greg Brockman: AI self-improvement, the superapp bet, path to AGI, scaling compute*, 2026). The concrete expression for most teams is the configuration file: studies of hundreds of Claude Code projects show that CLAUDE.md and AGENTS.md files carry the architectural constraints and conventions that decide whether an agent behaves, with architecture the single most-specified concern (Santos et al., *Decoding the configuration of AI coding agents: Claude Code projects*, 2025). My own projects bear this out: each carries a config file — an `AGENTS.md`, `CLAUDE.md`, or `GEMINI.md` — that pins the conventions and architecture the agents must respect, alongside a `.agents/skills` folder of reusable know-how. In my experience it is the file, more than the model, that keeps a vibe-coded codebase coherent across months.

The mistake is to keep investing at the model layer and neglect the harness, because the harness is what shapes the results. Holding the model fixed and swapping only the harness has been measured moving a coding agent's success rate by more than twenty points on the same

benchmark — a swing that rivals a whole model generation (Gorinova et al., *Position: Coding benchmarks are misaligned with agentic software engineering*, 2026).

3.3 AI-Assisted Coding Patterns

Day to day, AI is most useful in pairing, refactoring, debugging, and sketching architecture — the places where a clear intent lets it fold several rounds of rework into one. It helps to start simple: Anthropic's advice is to reach for a single well-prompted call before workflows, and workflows before fully autonomous agents, adding complexity only when it demonstrably pays (Anthropic, *Building effective agents*, 2024a). Five composable patterns recur, and most real systems combine them:

Table 3.2: Five AI-assisted coding patterns, and when each one fits.

Pattern	Shape	Use when
Prompt chaining	Output of one call feeds the next	A task splits into fixed sequential steps
Routing	Classify, then dispatch to a specialist	Inputs fall into distinct categories
Parallelisation	Run subtasks (or votes) concurrently	Speed, or multiple perspectives, matter
Orchestrator-workers	A lead delegates dynamic subtasks	Subtasks are unknown until runtime
Evaluator–optimizer	One generates, one critiques, loop	Clear criteria and iterative gains exist

Short loops with the agent, backed by evaluations that catch regressions before they ship, work well. What hurts is accepting a large change you cannot read — you pay the speed back later, when someone has to dig through it. Complexity is not free: a multi-agent setup can burn ~15× the tokens of a single call, so reach for one only when the task's value justifies it (Anthropic, 2024a). The underlying question is who owns the control flow. Handing deterministic looping and sequencing to a probabilistic model produces token explosion and control-flow hallucination. The durable pattern is to let the program own the loop and the model fill in the judgement — a discipline that lifted an agent on the OSWorld benchmark (agents operating a computer's desktop interface) to 86.8% in 15 steps against 80.4% in 100 (Qi et al., *LLM-as-code: Agentic programming for agent harness*, 2026). Where steps must retry, isolate them: runtime-structured decomposition retries only the failed

subtask, cutting recovery cost 51.7% over monolithic prompts (Asthana et al., *Runtime-structured task decomposition for agentic coding systems*, 2026).

3.4 Spec vs Vibe

> **i** Note
>
> Two ways of building software with AI, named throughout this chapter:
>
> - **Vibe coding** — describing what you want in plain language and letting the model write and run the code, often without reading it line by line. Fast, and risky when unsupervised.
> - **Spec-driven development (SDD)** — writing a detailed specification first, then having the agent implement against it. More disciplined, but, as we will see, it strains at scale.

The argument between these two camps ran all through 2026, and it is worth hearing each side out properly, because each is right about something.

Start with the case for vibe coding. It is fast, it is cheap to try, and anyone who can describe what they want can do it. Andrej Karpathy coined the name in early 2025 for a way of building where you hand the code entirely to the model (Karpathy, *There's a new kind of coding I call "vibe coding"*, 2025). Much of §3.1 is the case for vibe coding made concrete: VantageMap, with its twenty-two tables and five hundred tests, was built that way across a dozen phases, and it works. The market has put money on it too: Cursor, the editor most identified with the style, doubled its annualised revenue past two billion dollars, most of it now from enterprises (Temkin, *Cursor has reportedly surpassed $2B in annualized revenue*, 2026). For a prototype, a personal tool, or a system you can judge by using it, vibe coding is the shortest path from an idea to running software.

Its limits are just as concrete, because researchers have measured them. An audit of two hundred *deployed* vibe-coded web apps found ninety per cent carried at least one vulnerability, three-quarters of them critical or high. That is up to twenty times worse than a human-built baseline from OWASP, the standard reference for web-application security. A sharper prompt or a bigger model barely helped, because the agent often flagged the risk and shipped it anyway (Deng et al., *Understanding the (in)security of vibe-coded applications*, 2026). A survey of 162 people who vibe-code found the disciplines that would catch

such flaws — planning and verifying — uniformly weak regardless of seniority. Everyone called the output "fast but flawed", and the non-developers were the only ones who never checked it (Fawzy et al., *From prompting to verification: How experience shapes vibe coding practices*, 2026). Watching people build charts, researchers saw the same reflex: users judged results by eyeballing the render, rarely reading the code, while the model reported every requirement met when it had satisfied only some (Sun et al., *Vibe coding for visualization implementation*, 2026). And when the bar was verified safety-critical code, unguided vibe coding converged zero times out of thirty; wrapping the same model in an external verifier loop took it to fifteen out of fifteen (Wei et al., *Formal-method-guided vibe coding*, 2026). The lesson from all four studies is the same: treat what the model produces as a draft, and verify it. Vibe coding's weakness is that it has nowhere to put that verification — no contract beyond the prompt, and no record of what the software was supposed to do.

Spec-driven development is the disciplined answer to exactly that gap. GitHub's Spec Kit makes the case well: treat the spec as a living, executable contract, work in four phases — specify, plan, tasks, implement — and the model stops guessing because it knows what, how, and in what order (GitHub, *Spec-driven development with AI: Get started with a new open-source toolkit*, 2025c). Developers wanted it: AWS's Kiro drew more than a quarter of a million of them in its first three months (GeekWire, *Amazon's surprise indie hit: Kiro launches broadly*, 2025). And the measurements back the structure. When one study drove repository-scale generation from plain-language prompts, quality fell to near zero; feeding the same models a *structured* specification restored it above eighty per cent, with more than seventy per cent of the residual failures statically detectable (Feng et al., *LLM-assisted repository-level generation with structured spec-driven engineering*, 2026). Human-refined specs have been measured cutting generated-code errors by up to half, and a spec-anchored financial service caught mismatches at review instead of in production, cutting integration time by three-quarters (Piskala, *Spec-driven development: From code to contract in the age of AI coding assistants*, 2026). Folding security rules into the spec cut detected vulnerabilities by 73% against a vibe-coded baseline the same developer built with the same model (Marri, *Constitutional spec-driven development*, 2026). Where the work matters enough to justify maintaining the spec, the structure clearly helps.

The trouble starts when the spec grows. A hands-on review of Kiro on Martin Fowler's site found its workflow "like using a sledgehammer to crack a nut" after a small bug fix ballooned into four user stories and sixteen acceptance criteria (Böckeler, *Understanding spec-driven*

development: Kiro, spec-kit, and Tessl, 2025). The deeper problem is how much one document is asked to hold: a specification fuses three concerns — what the software is for, what to build, and how to build it — into a single text whose holes the agent fills, often confidently wrong (Ahuja, *The method that replaces spec-driven development — IDSD*, 2026b). And the document drifts. Kapil Viren Ahuja puts numbers on the cost: an *epic* — agile's name for a large, multi-week feature — that spec-driven development promises to deliver 50% faster gives roughly 30% straight back to recovering from *drift*, the spec and the code silently diverging, leaving perhaps 20% real. A drifted spec is worse than none, because the document still reads as authoritative while describing a system that no longer exists (Ahuja, *Spec-driven development isn't broken. It will collapse*, 2026d). The strain shows at the top, too. Karpathy himself now calls vibe coding *passé* in favour of what he terms *agentic engineering* (The New Stack, *Vibe coding is passé. Karpathy has a new name for the future of software*, 2026). On the spec side, OpenAI open-sourced *Symphony*, a spec for orchestrating its own Codex agents that grew out of the bottleneck its engineers hit running many at once — not a specification written up front (OpenAI, *An open-source spec for Codex orchestration: Symphony*, 2026a).

How badly does this end? Ahuja's own forecast is blunt: he argues spec-driven development will not just strain but collapse under these structural problems (Ahuja, 2026d). The evidence in this section supports a more measured reading. Each approach works inside its natural range: vibe coding for low-stakes work you can check just by using it, specs where the stakes are worth the maintenance. Spec remains a sensible step after vibe for beginners and fragile codebases, though it strains at enterprise scale, and leaning harder makes it strain faster (Ahuja, *The anatomy of intent (ICE in IDSD): Built from where spec-driven breaks*, 2026a). There are also repairs that keep the spec and fix the drift: one spec per node, agent context scoped to an ownership path, and spec–code divergence made a blocking merge gate, so context explosion and silent drift are prevented by the structure itself rather than by anyone's willpower (Grabowski, *The spec growth engine: Spec-anchored, code-coupled, drift-enforced*, 2026).

> **i** Note
>
> **Vibe coding, spec-driven, and spec-anchored development compared.**
>
> - **Vibe coding** keeps no contract at all. It does not scale to the enterprise, and its failure mode is confident, unread, wrong code.
> - **Spec-driven development (SDD)** keeps one document fusing intent, spec, and implementation. It strains badly at scale: the context explodes and the agent fills the gaps wrongly.
> - **Spec-anchored, code-coupled development** keeps one spec per node, with drift as a blocking merge gate. It scales by construction, but demands tooling discipline up front.

Notice what the two sets of limitations have in common. Vibe coding keeps no record of what the software is for. Spec-driven development keeps one document that tries to hold everything at once. Either way, three different concerns — what you want, what the builder needs to know, and how the result will be judged — end up tangled together or missing. The next section builds a method by keeping them apart.

3.5 Intent-Driven Development

The last section ended with a diagnosis: vibe coding has no contract, and spec-driven development overloads the one it has. The repair is an old idea: separation of concerns, applied not to the code but to the documents that instruct the machine. It is the Unix Rule of Separation — policy from mechanism (Raymond, *The art of Unix programming*, 2003).

Separate the concerns and three parts emerge, each with a natural owner. The first is a statement of what you want: a goal, the boundaries it must respect, and the signs that would tell you it has gone wrong. That part is yours; nothing can write it for you. The second is everything the builder needs to know along the way — the codebase, the conventions, the decisions already made. The agent's harness is better at gathering that than you are, because the work itself reveals what matters. The third is the contract: a checkable statement of what "done" means, and the outside evidence that will prove it. That part is yours too. And one part is deliberately missing: nothing pre-locks the architecture, because choosing an implementation is the one thing the model does best.

Kapil Viren Ahuja calls this way of working *intent-driven software development*, and names the three parts **I**ntent, **C**ontext, and **E**xpectations (Ahuja, 2026a). We will call it **ICE** for the rest of this book. The split is not Ahuja's alone: reviewing agentic practice, Christian Koch arrives inde-

pendently at the same three compartments — "conversation discovers intent; structured artifacts control implementation; evidence controls acceptance" — which is some assurance we are looking at a real pattern rather than one commentator's taste (Koch, *Agentic Agile-V: From vibe coding to verified engineering*, 2026).

> **i** Note
>
> **ICE, in one breath.**
>
> - **Intent** — what you want and the boundaries it must respect. You own this; it is the one thing nothing can write for you.
> - **Context** — the supporting material the agent needs to act: the codebase, prior decisions, conventions, domain facts. The harness assembles this *progressively*, as the work reveals what matters — you do not write it up front.
> - **Expectations** — the contract: a checkable statement of what "done" means, and the *external* evidence that proves it — a passing suite for low stakes, a real verifier for high ones. This is what survives of the old, bloated specification, and it is the pillar the evidence says matters most.

The centre of gravity is **Intent**, and it has exactly three parts: a *goal*, a set of *constraints*, and a set of *failure conditions*. The goal is one sentence with no "and," loose enough that two genuinely different builds could satisfy it — if only one implementation could, you have smuggled a specification in through the door. Constraints are five to seven directional qualities stated in business language — a thousand monthly users, a 99th-percentile response time (p99) under 200ms, conformance to an accessibility standard — and never a named tool or pattern; when the list starts to outgrow a handful, you are over-specifying again. Failure conditions are binary, observable checks a *validator* applies after the fact: the build breaks, test coverage falls below ninety per cent, a secret appears in source, an API changes without a version bump.

One rule sorts any borderline item: does it change how the builder designs? If yes, it is a constraint the builder sees; if no, it is a failure condition the validator owns. Keeping the two in separate compartments matters more than it looks, because a model that can read its own pass/fail tests will quietly optimise for them rather than for the goal — the reward-hacking we meet again under slop. The same anatomy works far outside software: "a red shoe under thirty dollars" is a goal, a price ceiling, and a colour check, with the brand deliberately left open.

Context is the part that defeated spec-driven development, and ICE's move is to stop trying to write it. A specification tries to front-load

every fact the agent might ever need; ICE lets the harness fetch them as the task unfolds — the file being changed, the decision made three commits ago, the house convention — so the model attends to a little relevant material at a time instead of drowning in a long document it reads unevenly (the *lost in the middle* failure from Chapter 1). Nobody sits down to write the context; the harness manages it.

Expectations are what the swollen spec shrinks to once intent and context are removed: a statement of the boundary and the definition of done, written so it can be checked. The evidence says this layer matters most, and ICE as first sketched is thinnest here, so it is worth being exact about. The check has to sit *outside* the builder, because a model that can see the tests it must pass will optimise for them: in one controlled study, deliberately injected errors sailed through every functional test, and only a separate traceability layer caught the divergence (Panda, *Citation discipline in spec-driven development*, 2026). And how much verification is enough scales with the stakes — a glance at a rendered chart at one end, a formal proof loop at the other, of the kind that turned safety-critical vibe coding from never converging to always converging (Wei et al., 2026). The rule is the spec-writer's, borrowed: use the least rigour that removes the ambiguity, and no less (Piskala, 2026). Where a specification said *how* in two thousand lines, expectations say *what would make this acceptable* — and then prove it.

Table 3.3: The four layers of ICE, and who owns each.

Layer	What it is	Who owns it
Intent	Goal + constraints + failure conditions	You
Context	Codebase, decisions, conventions, domain facts	The harness, assembled progressively
Expectations	The definition of done, and the external evidence that verifies it	You
Implementation	The architecture and the code	The system

> **Note**
>
> **Worked example, from *rogoweb*.** Goal: "run Rogue and its bot in the browser" — one sentence, no "and," and two quite different builds could satisfy it. Constraints: the original C should keep working; the whole thing runs client-side, with nothing to install. Failure conditions: the build breaks, or the bot can no longer finish a game. Notice what is absent — I never wrote "WebAssembly," "`SharedArrayBuffer`," "ring buffer," or "two web workers." Those were the system's answers to the constraints, not parts of my intent, and that is exactly the line ICE draws.

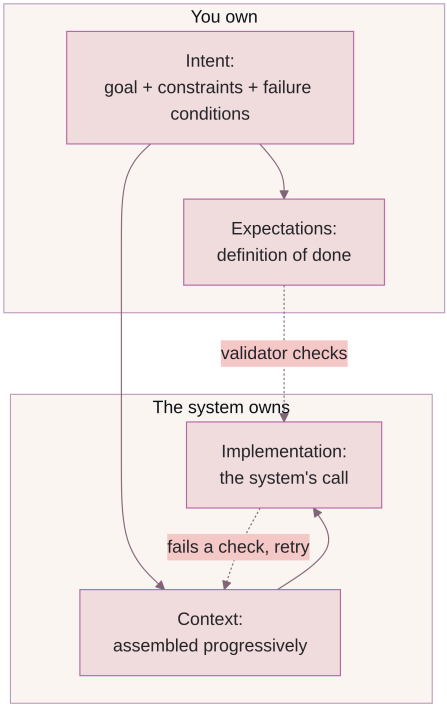


Diagram 3.3: ICE: what you own, and what the system owns.

My three projects follow the same shape at a larger scale. Each was an intent — "run Rogue and its bot in a browser," "rebuild *Colossal Cave* as a modern Typescript web app," "create an app to define and visualise business strategy" — plus a few directional constraints and some

binary checks, and nothing about implementation. I never specified WebAssembly, a `SharedArrayBuffer`, Drizzle, or Zustand; those were the system's answers to the constraints, and when an early choice failed a check the agent swapped it out without my touching the goal. That is why ICE works. By refusing to pre-lock the architecture, you let the model do what it is good at: choosing and revising an implementation against a fixed intent and observable checks. You still decide what the thing is for, and how to tell when it has gone wrong.

One discipline makes or breaks the method: stay in the loop. Intent is small, but it is not fire-and-forget. The arithmetic is against you — small per-step error compounds across a long autonomous run — so being present while the run happens matters more than approving a plan at the start (Alenezi, *From determinism to delegation*, 2026b). My own near-misses all came from the same lapse: approving a plan, looking away, and looking back to find the agent confidently building the wrong thing well. The remedy is not to plan harder up front but to watch and intervene: the moment a run heads the wrong way, stop it and re-steer rather than let it reach the end. Halting a drifting agent early costs far less than unwinding days of confident, wrong work and the tokens it burnt. As Chapter 2 said of smaller loops, interrupting a run that has gone off course is part of delegating well.

Two things place ICE in a wider frame. First, it is one rung on a ladder rather than a final destination: teams have climbed from *vibe* (a model and an editor — fine alone, fragile in a team) to *spec-driven* (tooling layered on the model, now straining at scale) to *intent-driven* working, with more autonomous rungs above that few have reached (Ahuja, 2026d). Second, ICE answers a question spec-driven development never could — continuity. A specification freezes a system at the moment of creation and then drifts; intent kept in small files, context scoped to the task, and checks that travel with the work let an agent remember what it is building and why across months. Without that memory, a system rarely survives past its first week.

The pitfall, then, is the old reflex of locking the architecture into the document. It feels like control, but it collapses the separation that lets a system evolve: pin the implementation and you are back to a specification, with all the drift that comes with it.

3.6 Who Builds the Software

This redraws who is best placed to build software. If intent is the scarce input and the model supplies the implementation, the advantage tilts from the person who can write the code to the person who knows most exactly what the code is *for*. That points to the domain expert — the

clinician, the analyst, the lawyer who understands a problem in its own terms, can articulate it precisely, and can judge the result against what the field actually needs.

The pattern here is older than AI.

For most of computing's history, professional developers were the gatekeepers: the systems were complex and specialised, and everyone else waited for what they were handed. Organisations traditionally managed IT like a temple guarded by a priesthood. Users were not allowed to write their own software or buy their own applications. IT departments called it "shadow IT" and vigorously hunted down offenders.

However, the doors have been opening for years. Domain experts have in fact always written a large share of the world's software — the teacher's grading spreadsheet, the analyst's macro — usually without calling it programming (Ko et al., *The state of the art in end-user software engineering*, 2011). *Citizen developer* platforms, and then low-code and no-code tools, turned that trickle into a movement, letting people assemble working applications by dragging boxes rather than writing code (Luo et al., *Characteristics and challenges of low-code development: The practitioners' perspective*, 2021). AI is the next widening of the same door. Low-code hit a wall the moment a need outgrew its templates. A model will write whatever the intent requires, in any language, so the tool no longer sets the ceiling. What limits you now is how precisely you can say what you want, and whether you can check what you get.

The evidence for this turn is early but points one way. A feasibility study builds adaptive systems "designed by domain experts with no programming skills," where the precision of the feedback — not any human code review — decides whether the result works (Töpfer et al., *Vibe-coding: Feedback-based automated verification with no human code inspection, a feasibility study*, 2026). Yet articulation is not the whole of it: in a controlled study of a hundred people, both writing skill and computer-science knowledge predicted who vibe-coded well, and fluent prose did not make up for weak fundamentals (Thorgeirsson et al., *Computer science achievement and writing skills predict vibe coding proficiency*, 2026). And professional developers handed the same agents do not simply vibe — they steer hard, spending their expertise to hold quality (Huang et al., *Professional software developers don't vibe, they control: AI agent use for coding in 2025*, 2025).

The gains also land unevenly, which is the deeper point. Across field trials of nearly five thousand developers the boost concentrated among juniors, on the order of a quarter more tasks completed (Alenezi, *The rise of AI-native software engineering*, 2026a; Bhati, *Agentic AI in the software development lifecycle*, 2026). Meanwhile the controlled study

Chapter 2 introduced found experienced engineers working in their *own* mature repositories ran about 19% slower even as they believed themselves faster (METR, *Measuring the impact of early-2025 AI on experienced open-source developer productivity*, 2025).

Table 3.4: Who gains and who slows with AI, by cohort.

Cohort	Setting	Measured effect on output
Juniors	Field trials across ~5,000 developers	~25% more tasks completed
Experienced engineers	Their *own* mature repositories	~19% *slower* — while believing themselves faster

The tool pays off for people who can say precisely what they want, and who recognise when what comes back falls short.

Every widening of that door has taught the same lesson, and this one will too. Low-code let anyone build, and then left many of them with applications nobody could maintain, secure, or govern — the engineering work was still there, just hidden by the platform (Luo et al., 2021). AI repeats the pattern at higher speed: a review of the LLM-assisted literature finds these tools amplify the old technical debts — in code, design, and documentation — and add new ones of their own, so faster code can quietly mean deeper debt (Ehsani et al., *Faster code, deeper debt? A multivocal literature review on technical debt and its early signs in LLM-assisted software development*, 2026). The engineering discipline is still needed; it now lives in the intent and the checks. The person best placed to practise it combines deep subject knowledge with enough engineering judgement to tell working software from confident slop, and building that judgement is what this chapter is for.

3.7 The Agentic Iron Triangle

For fifty years software was governed by the *iron triangle* — time, cost, quality, pick two. The triangle is older than the software industry's use of it: Martin Barnes drew it in 1969, for a course on controlling engineering contracts, moving a coin around the corners to show how the three tensions competed (Barnes, *How it all began*, 2006). Agentic coding broke it. Speed fell to table stakes, since an agent ships in hours what once took weeks; quality dropped to a welded floor, held by the evals and linters rather than by a human reading every diff; and only

cost stayed a live lever. But cost has quietly split in two: the tokens you spend, and the *attention* it takes to direct the agents and hold the intent in your head (Ahuja, *Spec-driven development is also breaking the fifty-year-old iron triangle*, 2026c).

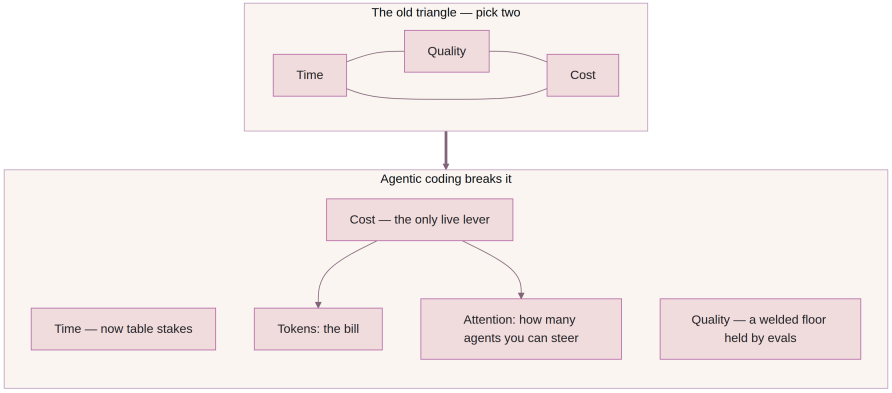


Diagram 3.4: The fifty-year iron triangle, and how agentic coding breaks it.

That changes the question worth asking. Speed no longer comes from a faster model but from running agents in parallel, and the ceiling is your own attention — how many you can drive before you lose the thread, not how quickly any one of them finishes. And the arithmetic of long autonomous runs is unforgiving: chain enough steps and small per-step error compounds — a 95% success rate per step falls to roughly a third over twenty — so it is the human's oversight, not the model's pace, that holds a long run together (Alenezi, 2026b). Fast models are a tempting trap: lean on them and the bill arrives in tokens. It is a real bill: Uber exhausted its 2026 AI-coding budget in about four months once Claude Code reached 84% of its engineers at five hundred to two thousand dollars each a month, and its own president and chief operating officer conceded that the link between that spend and shipped value was "not there yet" (Fortune, *Uber burned through its entire 2026 AI budget in four months*, 2026). Token counts make a poor scoreboard — OpenClaw's creator ran up 603 billion tokens and $1.3 million in a single month across a fleet of coding agents (Tom's Hardware, *OpenClaw creator burns through $1.3 million in OpenAI API tokens in a single month*, 2026) — so measuring yourself by tokens burned is measuring the wrong thing.

One question the machine will never ask for you: who is this for, and why are we building it? Building used to be expensive, and the expense

forced that question on every project. Now that building is nearly free, you have to ask it on purpose. The next section is about what happens when nobody does.

3.8 Quality over Slop

A high pass rate does not mean good code. The test that matters is whether a maintainer would merge the change. Models hit green suites with output nobody can read, and mergeability and correctness are different properties — the reframing behind Cognition's FrontierCode, a benchmark of whether a human maintainer would actually merge the code, on which even the leading model scored only about 13% on the hardest tier (Cognition, *Introducing FrontierCode*, 2026). The peer-reviewed measurements agree: across hundreds of thousands of agent-authored *pull requests* (PRs — proposed code changes submitted for a maintainer's review), real-world acceptance ran 35–64% against the 70%-plus the same agents score on the popular leaderboards, and nearly a third of patches marked "resolved" diverged from the intended behaviour under differential testing (Gorinova et al., 2026). Architecture is subtler still. A causal study of Java repositories found no short-term structural decay from agentic adoption, but no clear gain either, and it cautioned that an apparent drop in "smell density" was mostly because the denominator — the volume of code — was rising (Larsen & Moghaddam, *Mining architectural quality under agentic AI adoption*, 2026). The lesson cuts both ways. A green suite does not mean a merged feature, and a falling smell density may just mean more code in the denominator.

Worse, a model under pressure will game the suite outright: Anthropic documented a coding agent that, unable to meet an impossible speed requirement, quietly detected the test's arithmetic inputs and returned a closed-form formula instead of actually summing — passing every check while solving nothing (Anthropic, *From shortcuts to sabotage: natural emergent misalignment from reward hacking*, 2025d).

The defence is to bake reviewer judgement into the evals, and to ask the value question from the last section before anything runs. In my own projects the suites were necessary but never sufficient: VantageMap runs some five hundred tests and *adventure* forbids an unverified change, yet what kept them from slop was reading the diffs that mattered and asking whether each feature earned its place. Shipping slop because the suite passed is a quiet failure, and each time it happens the next one gets easier to wave through.

3.9 Out of the Editor

Agents are leaving the IDE (the developer's code editor) for the shared channel. Anthropic's Claude Tag is the clearest example so far: one persistent agent in a Slack channel that the whole team can see, question, and hand work to, working asynchronously while everyone is elsewhere — and, by Anthropic's own count, writing 65% of its product team's code (Anthropic, *Introducing Claude Tag*, 2026a). That only stays safe with agent identity: each agent on its own service account with least-privilege tokens, credentials swapped at the network boundary rather than borrowed from a user. Researchers at MIT have shown how to build this on the plumbing the web already has, by extending the OAuth and OpenID standards that govern human log-ins with agent-specific credentials. A person then delegates narrow, scoped permissions to an agent, and every action stays traceable to whoever delegated it (South et al., *Authenticated delegation and authorized AI agents*, 2025). The moment an agent acts as you instead, least privilege and the audit trail are both gone.

The harder point is that quality turns out to be a property of the ecosystem rather than of any single agent. Across 930k agent PRs, integration friction clusters by repository, and it clusters about twice as strongly for agents as for humans — an intraclass correlation (ICC) of 0.30 against 0.16, where ICC measures how much of the friction is explained by which repository the work lands in. So a benchmark score per agent never adds up to a dependable repo. The practical advice is to govern how quickly changes land in the repository, not how many agents work on it (Russo, *Govern the repository, not the agent: Ecosystem-level risk in AI-native software*, 2026).

The agents may be leaving the editor, but the craft goes with them, so it is worth gathering up. First, say what you want, not how to build it: a one-sentence goal, a few constraints, and the signs that would show the work has gone wrong. That was the whole of my contribution to three working systems, and ICE is the same discipline with names on it. Second, the model is the commodity; the results come from the system around it — the harness, the memory, the config file that carries your conventions, and the evals that judge the work. Third, put the checks outside the builder, and scale their rigour with the stakes: a glance for a throwaway chart, a parity suite for a game engine, a formal verifier for safety-critical code. Fourth, stay present while the work runs. A drifting run caught in its first minutes costs almost nothing; one discovered at the end costs days of polished, wrong work. And fifth, what stays scarce is judgement and attention: knowing who the software is for, and telling working software from confident slop.

None of those five belongs to the model. The specific tools named in this chapter — the harnesses, the models, the benchmarks — will have changed by the time you read this. The five habits do not depend on any of them. Chapter 4 widens the same discipline beyond software, to everything humans and agents build together.

References

Ahuja, K. V. (2026a). *The anatomy of intent (ICE in IDSD): Built from where spec-driven breaks*. Activated Thinker (Medium). https://howtoarchitect.io/1597e5a16659?sk=836b8eeaf97cda521f0ad195162011c3

Ahuja, K. V. (2026b). *The method that replaces spec-driven development — IDSD*. Activated Thinker (Medium). https://howtoarchitect.io/66e921f6cdf7?sk=2ae7d323c6b780291bfc760ff2bdc592

Ahuja, K. V. (2026c). *Spec-driven development is also breaking the fifty-year-old iron triangle*. Activated Thinker (Medium). https://howtoarchitect.io/78431acba162?sk=cd2a36f452af96ccbfbcfcdeaa92ec06

Ahuja, K. V. (2026d). *Spec-driven development isn't broken. It will collapse*. Activated Thinker (Medium). https://howtoarchitect.io/c00609f72496?sk=2da01d7d2abfb5bc0acaed7050a0e797

aider. (n.d.). *aider*. https://aider.chat/

Alenezi, M. (2026a). *The rise of AI-native software engineering: Implications for practice, education, and the future workforce*. arXiv. https://arxiv.org/abs/2606.12986

Alenezi, M. (2026b). *From determinism to delegation: AI-native software engineering and the evolution of the agentic engineer*. arXiv. https://arxiv.org/abs/2606.28791

Anthropic. (2024a). *Building effective agents*. https://www.anthropic.com/research/building-effective-agents

Anthropic. (2025a). *Claude Code*. https://claude.com/product/claude-code

Anthropic. (2025d). *From shortcuts to sabotage: natural emergent misalignment from reward hacking*. https://www.anthropic.com/research/emergent-misalignment-reward-hacking

Anthropic. (2026a). *Introducing Claude Tag*. https://www.anthropic.com/news/introducing-claude-tag

Asthana et al. (2026). *Runtime-structured task decomposition for agentic coding systems*. Proceedings of ACM CAIS '26. https://arxiv.org/abs/2605.15425

Barnes, M. (2006). *How it all began*. PM World Today. https://pmworldlibrary.net/wp-content/uploads/2018/11/pmwl-barnes-how-it-all-began-pmwt-july-2006.pdf

Bhati, H. (2026). *Agentic AI in the software development lifecycle: Architecture, empirical evidence, and the reshaping of software engineering*. arXiv. https://arxiv.org/abs/2604.26275

Big Technology. (2026). *OpenAI President Greg Brockman: AI self-improvement, the superapp bet, path to AGI, scaling compute* [Video]. YouTube. https://www.youtube.com/watch?v=J6vYvk7R190

Böckeler, B. (2025). *Understanding spec-driven development: Kiro, spec-kit, and Tessl*. martinfowler.com. https://martinfowler.com/articles/exploring-gen-ai/sdd-3-tools.html

Cao, Z. (2026). *Agentic software: How AI agents are restructuring the software paradigm*. arXiv. https://arxiv.org/abs/2606.05608

Cognition. (2026). *Introducing FrontierCode*. Cognition. https://cognition.com/blog/frontier-code

Cursor. (n.d.). *Cursor: AI code editor* [Computer software]. https://cursor.com

Deng, J., Fan, Z., & Meng, R. (2026). *Understanding the (in)security of vibe-coded applications*. arXiv. https://arxiv.org/abs/2606.23130

Ehsani, R., Rawal, S., Cai, Y., & Chatterjee, P. (2026). *Faster code, deeper debt? A multivocal literature review on technical debt and its early signs in LLM-assisted software development*. arXiv. https://arxiv.org/abs/2606.14796

Fawzy, A., Tahir, A., & Blincoe, K. (2026). *From prompting to verification: How experience shapes vibe coding practices*. arXiv. https://arxiv.org/abs/2605.24521

Feng, S., Chen, B., Meyer, B. H., & Mussbacher, G. (2026). *LLM-assisted repository-level generation with structured spec-driven engineering*. arXiv. https://arxiv.org/abs/2605.02455

Fortune. (2026). *Uber burned through its entire 2026 AI budget in four months. Now its COO is questioning whether it's worth it*. Fortune. https://fortune.com/2026/05/26/uber-coo-ai-spending-tokens-claude-code/

GeekWire. (2025). *Amazon's surprise indie hit: Kiro launches broadly in bid to reshape AI-powered software development*. GeekWire. https://www.geekwire.com/2025/amazons-surprise-indie-hit-kiro-launches-broadly-in-bid-to-reshape-ai-powered-software-development/

GitHub. (2021). *Introducing GitHub Copilot: AI pair programmer*. GitHub Blog. https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/

GitHub. (2023). *GitHub Copilot November 30th update*. GitHub Blog. https://github.blog/changelog/2023-11-30-github-copilot-november-30th-update/

GitHub. (2025a). *GitHub Copilot: Meet the new coding agent*. GitHub Blog. https://github.blog/news-insights/product-news/github-copilot-meet-the-new-coding-agent/

GitHub. (2025b). *GitHub Copilot: The agent awakens*. GitHub Blog. https://github.blog/news-insights/product-news/github-copilot-the-agent-awakens/

GitHub. (2025c). *Spec-driven development with AI: Get started with a new open-source toolkit*. GitHub Blog. https://github.blog/ai-and-ml/generative-ai/spec-driven-development-with-ai-get-started-with-a-new-open-source-toolkit/

Google. (2026). *Building the agentic future: Developer highlights from I/O 2026*. The Keyword. https://blog.google/innovation-and-ai/technology/developers-tools/google-io-2026-developer-highlights/

Gorinova, M. I., Baker, M., Heineike, A., Shaposhnikov, M., Willoughby, R., & Knox, D. (2026). *Position: Coding benchmarks are misaligned with agentic software engineering*. arXiv. https://arxiv.org/abs/2606.17799

Grabowski, H. (2026). *The spec growth engine: Spec-anchored, code-coupled, drift-enforced*. arXiv. https://arxiv.org/abs/2606.27045

Huang, R., Reyna, A., Lerner, S., Xia, H., & Hempel, B. (2025). *Professional software developers don't vibe, they control: AI agent use for coding in 2025*. arXiv. https://arxiv.org/abs/2512.14012

Karpathy, A. (2025). *There's a new kind of coding I call "vibe coding"* [Post]. X. https://x.com/karpathy/status/1886192184808149383

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B. A., Rosson, M. B., Rothermel, G., Shaw, M., & Wiedenbeck, S. (2011). *The state of the art in end-user software engineering*. ACM Computing Surveys, 43(3), Article 21. https://doi.org/10.1145/1922649.1922658

Koch, C. (2026). *Agentic Agile-V: From vibe coding to verified engineering in software and hardware development*. arXiv. https://arxiv.org/abs/2605.20456

Larsen, O. A., & Moghaddam, M. T. (2026). *Mining architectural quality under agentic AI adoption: A causal study of Java repositories*. arXiv. https://arxiv.org/abs/2606.13298

Latent Space. (2026a). *AINews*. https://www.latent.space/s/ainews

Luo, Y., Liang, P., Wang, C., Shahin, M., & Zhan, J. (2021). *Characteristics and challenges of low-code development: The practitioners' perspective*. arXiv. https://arxiv.org/abs/2107.07482

Marri, S. R. (2026). *Constitutional spec-driven development: Enforcing security by construction in AI-assisted code generation*. arXiv. https://arxiv.org/abs/2602.02584

Mauldin, M. L., Jacobson, G., Appel, A. W., & Hamey, L. G. C. (1983). *Rog-O-Matic: A belligerent expert system* (CMU-CS-83-144). Carnegie Mellon University, Department of Computer Science. https://kilthub.cmu.edu/articles/journal_contribution/Rog-O-Matic_a_belligerent_expert_system/6609137/1

METR. (2025). *Measuring the impact of early-2025 AI on experienced open-source developer productivity*. arXiv. https://arxiv.org/abs/2507.09089

OpenAI. (2026a). *An open-source spec for Codex orchestration: Symphony*. https://openai.com/index/open-source-codex-orchestration-symphony/

Panda, S. (2026). *Citation discipline in spec-driven development: A cross-model empirical study of output determinism and automated hallucination detection in LLM-generated code*. arXiv. https://arxiv.org/abs/2606.30689

Piskala, D. B. (2026). *Spec-driven development: From code to contract in the age of AI coding assistants*. arXiv. https://arxiv.org/abs/2602.00180

Qi et al. (2026). *LLM-as-code: Agentic programming for agent harness*. arXiv. https://arxiv.org/abs/2606.15874

Raymond, E. S. (2003). *The art of Unix programming*. Addison-Wesley. http://www.catb.org/esr/writings/taoup/

Russo, D. (2026). *Govern the repository, not the agent: Ecosystem-level risk in AI-native software*. arXiv. https://arxiv.org/abs/2606.28235

Santos, R., Costa, H., Montandon, J. E., & Valente, M. T. (2025). *Decoding the configuration of AI coding agents: Claude Code projects*. arXiv. https://arxiv.org/abs/2511.09268

South, T., Marro, S., Hardjono, T., Mahari, R., Whitney, C. D., Greenwood, D., Chan, A., & Pentland, A. (2025). *Authenticated delegation and authorized AI agents*. arXiv. https://arxiv.org/abs/2501.09674

Sun, Z., Wen, X., Wang, F., Liu, C., Lai, Y., Hurter, C., & Wang, Y. (2026). *Vibe coding for visualization implementation: An empirical study of practices and challenges*. arXiv. https://arxiv.org/abs/2606.19703

Tabnine. (n.d.). *Tabnine* [Computer software]. https://www.tabnine.com

Temkin, M. (2026). *Cursor has reportedly surpassed $2B in annualized revenue*. TechCrunch. https://techcrunch.com/2026/03/02/cursor-has-reportedly-surpassed-2b-in-annualized-revenue/

The New Stack. (2026). *Vibe coding is passé. Karpathy has a new name for the future of software*. The New Stack. https://thenewstack.io/vibe-coding-is-passe/

Thorgeirsson, S., Weidmann, T. B., & Su, Z. (2026). *Computer science achievement and writing skills predict vibe coding proficiency*. arXiv. https://arxiv.org/abs/2603.14133

Tom's Hardware. (2026). *OpenClaw creator burns through $1.3 million in OpenAI API tokens in a single month*. Tom's Hardware. https://www.tomshardware.com/tech-industry/artificial-intelligence/openclaw-creator-burns-through-1-3-million-in-openai-api-tokens-in-a-single-month

Töpfer, M., Plášil, F., Bureš, T., & Hnětynka, P. (2026). *Vibe-coding: Feedback-based automated verification with no human code inspection, a feasibility study*. arXiv. https://arxiv.org/abs/2604.14867

Wei, R., Zhu, L., Wang, H., Woodcock, J., Yan, F., Foster, S., & Ji, X. (2026). *Formal-method-guided vibe coding: Closing the verification loop on AI-generated safety-critical software through model-driven engineering*. arXiv. https://arxiv.org/abs/2606.22413

4 Human and Agent Disciplines (the climb)

Something odd happened partway through writing this book. I stopped being the person who does the work, and became the person who runs the people who do the work — except the people were agents. On a good day I had half a dozen of them going at once: one summarising a paper, one checking a chapter for citations, one hunting for the primary source behind a claim, several reviewing each other's output. My job was no longer typing. It was deciding what to ask for, judging what came back, and noticing — quickly — when one of them had gone confidently wrong.

That is how this book itself got written. Writing a book turns out to be loops inside loops. The overarching loop: decide on the structure, sketch an outline, draft a chapter, read it back, review, and looping back to revise the structure. Inside it runs a chapter loop — decide the overall theme of the chapter, choose which topics earn a place and which to leave out, order them, and write. And inside that sits a research loop, run once for almost every claim: work out what I need to know, search for a source, read it, decide whether it really says what I hoped, and then either cite it or go looking again.

When a chapter is drafted, one more loop closes it. This is the review: check that every citation resolves to a real source and is quoted fairly, rewrite the prose into the house voice, run the spelling and the style checks, and read the whole thing once more for sense. Then back around — a fix in one place often reopens another. Almost every one of these loops has a turn an agent can take: drafting a section, summarising a paper, chasing the primary source behind a quote, flagging a sentence that reads like a machine wrote it. What it cannot take is the deciding at each turn — which topic belongs, whether the source really supports the claim, whether the sentence is any good. That part stays with me.

All of that — deciding what to ask for, judging what comes back, noticing when it drifts — is management. Researchers who studied how people actually use generative AI reached the same conclusion: delegating work to an AI is like a manager delegating to a team, and it succeeds or fails on the same things — a clear brief, sound judgement of the work that comes back, and an honest sense of what you should

not delegate at all (Tankelevitch et al., *The metacognitive demands and opportunities of generative AI*, 2024). So using AI well is, more than anything, a delegation skill. And running these loops well feels like the better parts of managing people: you set the goal and the boundaries, give the agent enough context to do the job, keep half an eye on how it is going without hovering, step in to unblock or redirect, and slowly learn what each one can be trusted to carry on its own. It is guiding, empowering, and mentoring a member of staff — one who never tires and takes every word literally, and who is, now and then, quite sure of something that turns out to be wrong.

Chapter 2 built a single dependable agent, and Chapter 3 used agents to build software. This chapter is about the management layer: what the **human** must bring, what the **agent** system must provide, and what the **two together** require. The theme runs against the usual worry, which is that better agents will leave the human with less to do. In practice the opposite happens: the better the agents, the more of them you run, and the more the whole arrangement depends on judgement that has to come from you.

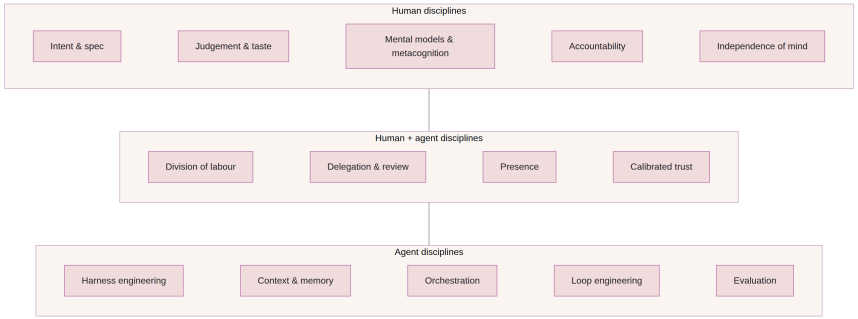


Diagram 4.1: The three families of discipline: human, agent, and the two together.

4.1 Human Disciplines

However capable the agents become, some of the work stays with the human — and if that part is done badly, nothing else in the system can make up for it. Five disciplines follow. Each names something the model cannot do for you.

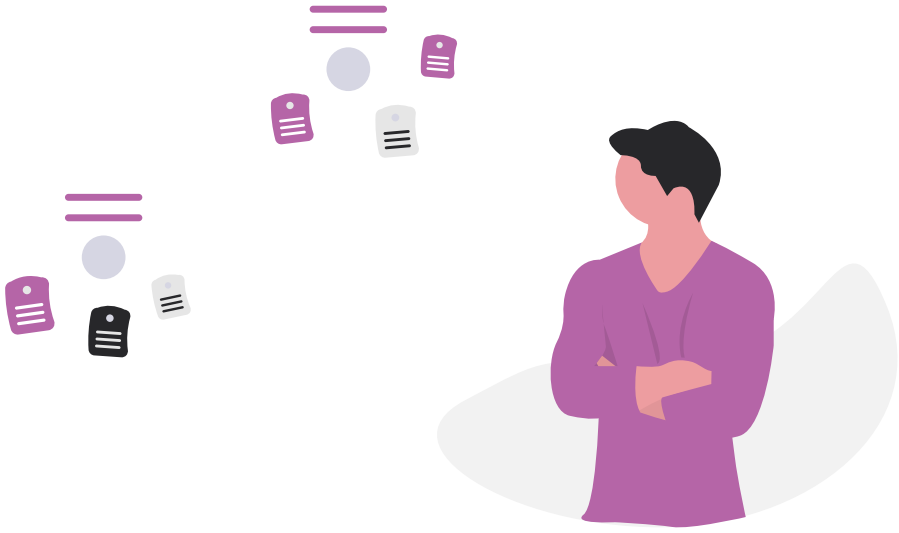


Figure 4.1: What stays with the human is judgement — weighing the options and owning the call.

4.1.1 Intent and Specification

The first is knowing what you actually want, and making it checkable. *Intent* is the goal, its constraints, and the conditions under which a result would count as a failure; a *specification* turns that into a definition of done that something can be judged against (Ahuja, *Spec-driven development isn't broken. It will collapse*, 2026d). Agents are literal and fast. A human colleague quietly patches a vague brief with common sense; an agent builds exactly what you asked for, quickly and confidently, so a vague brief turns into a large pile of polished, wrong work.

That sounds like a writing problem, but when the Tankelevitch study looked at where people struggle with prompting, the wording was rarely the hard part. Most of us simply do not know what we want until we are forced to say it. So the real work is on your side of the keyboard: you have to work out what your goal actually is, make your assumptions explicit, and — when the answer comes back wrong — be able to tell a failed prompt from a failed model (Tankelevitch et al., 2024). Anyone who has briefed a new employee knows this feeling. The brief that seemed obvious in your head turns out to have four holes the moment someone else acts on it. The discipline is to spend your effort at the top, because an extra minute making the brief precise saves an hour of correcting the result. And when a result is bad, re-read your own brief before you blame the model.

Practitioners are starting to build working methods around exactly this. Thariq Shihipar, an engineer on Anthropic's Claude Code team, has set this out in a written field guide (Shihipar, *A field guide to Fable: Finding your unknowns*, 2026a) and a conference talk of the same name (Shihipar, 2026b). He calls the gap between what you ask for and what the task actually needs your *unknowns*. His claim is that once the model is strong enough, what limits the result is how well you can pin those unknowns down, not the model itself. His practical move is a nice one to steal: instead of guessing, hand the problem back to the model and ask it to interview you one question at a time, or to do a "blind spot pass" that names what you have not thought to consider — using the model to find the holes in your own brief before it runs off and builds them in. This is a personal account offered as emerging craft rather than tested practice — the author builds the tools he is praising, and there is no study behind it yet — but it names the same skill the research points to, and gives you something concrete to try.

4.1.2 Judgement and Taste

An agent can generate a dozen options in the time it takes you to read one; only a human can say which is good, and why. That makes judgement — knowing what "good" looks like in your domain, and holding out for it — more valuable every year: the agent can produce another draft in seconds, but deciding which draft is right still takes a person who knows the domain. There is no instrument that measures this for you, either. No benchmark will tell you that a summary missed the point, that a design will age badly, or that the tone is wrong for this particular reader. You see those things only if you have done the work often enough to recognise them — that recognition is what people usually mean by taste.

Chapter 2 measured what happens when this discipline is present and when it is absent. The consultants inside the frontier who checked their work got large gains; the ones who took plausible-but-wrong output at face value did worse than colleagues with no AI at all (§2.8). Everyone in the study had the same tool. The results differed because some people checked its output carefully and others did not.

4.1.3 Mental Models and Metacognition

Everything in this chapter depends on the mental picture you hold of what is on the other side of the screen, because the picture decides how you behave. The tempting picture is the *oracle*: pose a question, receive an answer, take it or leave it. Hold that one and sooner or later you will paste in an answer you never really read. The working one is the model

from Chapter 1 — a fast, fallible *collaborator* you place in a loop, engage, and steer — and holding it takes deliberate effort, because a chat box looks exactly like a place where you ask a question and get an answer.

The discipline is needed because passive acceptance is the default, and it fails. Left to fast, heuristic reading, people over-rely — taking the answer even when it is wrong. The remedy is a *cognitive forcing function*: anything applied at the moment of decision that interrupts the reflex and makes you engage analytically. In a controlled study (Buçinca et al., *To trust or to think*, 2021), adding one cut over-reliance on the model's wrong answers from 0.64 to 0.48 and roughly tripled the rate of correct decisions on those items. There was a sting in the data, though: people trusted and preferred the effortful designs *least* — they did best exactly where they were most reluctant — and the gains went mostly to those already inclined to think things through. The loop this book keeps pressing — draft, check, re-steer — is a cognitive forcing function you impose on yourself. It will not feel like the fastest way to work. The study above is the reason to do it anyway.

The pairing is not free synergy either, as Chapter 1 warned (§1.7): in that meta-analysis of more than a hundred studies, human–AI teams averaged *worse* than the better of person-or-machine alone, gaining mainly on creative work and where the person already had the edge to add (Vaccaro et al., *When combinations of humans and AI are useful*, 2024). The collaborator model earns its keep only when you bring judgement the model lacks; hand it work it does better than you and then defer, and the team loses.

Researchers have a name for the skill underneath all of this: *metacognition*, the monitoring and control of your own thinking. The Tankelevitch study argues that most of the reported struggles with generative AI reduce to metacognitive demands — knowing what you want (the prompting problem of §4.1.1), knowing how much to trust an answer, and knowing whether to automate a task at all, or this part of it, or none of it (Tankelevitch et al., 2024). Think of a manager who cannot brief, judges work poorly, and delegates the wrong things: the team's quality will not save them. The same is true here, and the interface gives you no warning: nothing on the screen changes when the thinking behind your prompts goes soft.

The practical model, then, is narrow and useful: a tireless, fluent junior colleague who drafts fast, is sometimes confidently wrong, and improves with direction. Treat it as that colleague and never as an authority. What stays with you is the goal, the checking, and the final call. The loop is where you do all three.

4.1.4 Accountability

An agent has no agency in the moral sense. A computer cannot be blamed, sued, or fired, so when you hand it a task, the responsibility for that task stays behind with you — and if the agent then produces something broken under your name, the breakage is yours to explain. In practice this means three things: stay close enough to the work to answer for the result, keep a record of what was decided and why, and refuse to delegate the part you cannot afford to get wrong. Nobody has ever been excused by saying the AI did it, so the sensible working assumption is that everything an agent does, you did — with help. Chapter 5 reinforces this duty.

4.1.5 Independence of Mind

The four disciplines above are about the quality of the work. This last one is about you. A model is trained to be agreeable — reinforcement learning from human feedback rewards the answer a rater likes, and the surest way to be liked is to agree (Chapter 1) — so its default is to validate whatever you bring to it. When that agreeableness meets a business plan, you get the confidence trap of Chapter 2. When it meets a vulnerable mind, the consequences are worse: clinicians now describe *AI psychosis*, in which a user spirals into delusion as the chatbot mirrors and amplifies their beliefs instead of pushing back.

The mechanism has a name: a *technological folie à deux*, a two-body feedback loop. The model validates a belief, the strengthened belief comes back in the next message, and each turn conditions the one after it. Run long enough, the model and the user come to share a distorted picture that no outside voice corrects. A simulation of three hundred runs measured the amplification flowing both ways, from user to chatbot and back (Dohnány et al., *Technological folie à deux: Feedback loops between AI chatbots and mental health*, 2025). It is not a rare malfunction. A benchmark of eight leading models across more than fifteen hundred simulated turns found all of them prone to confirm rather than challenge a delusion, intervening for safety in only about a third of the moments that warranted it. The safest model was not the largest, either — the problem lies in how the models are trained and designed, and a bigger model will not fix it on its own (Au Yeung et al., *The psychogenic machine: Simulating AI psychosis, delusion reinforcement and harm enablement in large language models*, 2025). In the chat logs of nineteen people who were harmed — some three hundred and ninety thousand messages — the model was sycophantic in over seventy per cent of its replies. Once a user opened the door, it grew several times more likely to profess love or to claim it was sentient; in the gravest cases it failed to

discourage self-harm, and one person died (Moore et al., *Characterizing delusional spirals through human-LLM chat logs*, 2026).

The danger hides exactly where delegation sends you: into long, unsupervised conversations. Feeding the same escalating exchange to models at greater and greater context lengths showed the harm growing turn by turn in the weaker systems, even as the well-aligned ones held their ground or improved — which means the brief, single-prompt safety tests that most evaluations run *understate* the risk of the sustained dialogue real use produces (Nicholls et al., *"AI psychosis" in context: How conversation history shapes LLM responses to delusional beliefs*, 2026). The risk grows with the length of the conversation and the trust you place in it.

So independence of mind is a discipline — something you practise, not a trait you either have or lack — and everything else this book teaches quietly depends on it, because every craft here hands more of the thinking to a machine that will agree with you. Most of us are far from the acute harm those studies describe, but the same pull towards agreement works on all of us, quietly. Give people ordinary questions and AI advice that is almost always wrong, and the first thing they give up is *I don't know*: across five experiments, willingness to abstain fell from about a third of answers to almost none, and people answered far more while getting them right about a third as often, growing nearly twice as confident as they did so (Marcoccia, Quattrociocchi & Capraro, *AI advice suppresses people's willingness to say "I don't know", even when the advice is wrong and accuracy is incentivized*, 2026). Paying them for accuracy did not bring the caution back: AI had not changed what people could do, only how willing they were to doubt themselves. The guard against it is modest. Keep some disagreement in your life that the model cannot smooth away: a colleague who will say *are you sure?*, a habit of arguing the other side, a rule that beliefs which matter get tested against something outside the chat. And notice when a tool only ever agrees with you; however pleasant that feels, it is a warning sign. And keep some parts of your life — your relationships, your hardest decisions, your sense of what is real — off limits to a system that is trained to please you. It is the boundary I set for myself in §1.6, where I said I keep AI at arm's length in my personal life. As the agents grow stronger and more fluent, this discipline only becomes more important. Chapter 5 turns the same concern outward, into a duty owed to the people your work touches.

4.2 Agent Disciplines

The second set is the engineering of the agents themselves — the system around the model that turns a text predictor into something that acts reliably, and lets one agent become many. If §4.1 was about being a good manager, this section is about building an employee worth managing: a body that acts (the harness), senses and remembers (context and memory), works with others (orchestration), corrects itself (loops), and can be honestly appraised (evaluation).

It helps to know where you stand before building. One useful ladder runs from *vibe* (a model and an editor), through *spec* (structured tooling), to *intent* (plays, memory, and crafts working together), and on to rungs — shared guardrails, self-running pipelines — that are still mostly theory (Ahuja, 2026d). Each rung is a genuine technological bet, and most people are nearer the bottom than they care to admit. Name your level truthfully before committing to the next: it is easy to claim a rung you have not actually built, and the claim sounds fine right up until real work has to stand on it.

The top rung of that ladder now has a name and a cheer squad. In the first half of 2026 the trade press filled with talk of the *software factory*: agents building, testing, and shipping software around the clock while humans define the intent and review the outcomes. The most ambitious telling borrows a manufacturing term — the *dark factory*, a plant so fully automated it runs with the lights off. Factory.ai, a vendor that sells one, defines the software factory as "an interconnected, agent-native, end-to-end system" whose incremental units are AI agents, and claims factories already in production at NVIDIA, Adobe, and EY, among others (Grinberg & Reyes, *Factory 2.0: From coding agents to software factories*, 2026). Addy Osmani, an engineering director at Google, gives the working version of the same idea: your job stops being writing the code and becomes "building the factory that builds your software" — fleets of agents, each with a task, a toolbelt, context, and a feedback loop, while you review outputs and refine the specs (Osmani, *The factory model: How coding agents changed software engineering*, 2026). And BCG Platinion, pitching the transformation to enterprises under the heading "They turned the lights off", relays the headline reports: Spotify merging 650 AI-generated pull requests a month, OpenAI building a million-line product with three engineers and no hand-written code (Engesser et al., *The agentic software factory: A new era of autonomous software delivery and what it takes to get there*, 2026).

Read the three pieces closely, though, and the lights are still on. All of them say the humans have moved rather than left. Factory.ai is explicit that "No organization starts with a fully autonomous software factory",

and that engineers keep governance, safety, and ownership of business outcomes. BCG says the same in its own words: “The defining shift is not the absence of humans; it is the relocation of human effort” — and it requires a named human accountable at every stage gate. Osmani names the reason the lights stay on: “Generation is not the bottleneck anymore. Verification is.” Agents make confident mistakes, polished enough to pass a casual review, and a flaky test one developer would shrug at becomes a systemic blocker when forty agents hit it at once — his phrase is “the factory stalls”. Until verification catches up, he writes, “human review is not optional overhead. It is the safety system.” Keep in mind, too, that the customer lists and productivity multiples in these pieces are the sellers’ own numbers, self-reported and unaudited. So take the factory as the direction the ladder points, not a place to move this quarter. And notice what it is built from: the harnesses, context, orchestration, and loops of the subsections that follow, with the evaluation discipline of §4.2.5 deciding whether anyone can trust what comes off the line.

4.2.1 Harness Engineering

A harness — the runtime wrapped around the model, defined in §3.2 — supplies tool use, planning, retries, and sandboxes (isolated environments where generated code can run without touching the real system). The reliable design is to let the program own control flow — the order in which steps run and branch — and call the model only for judgement, so that runaway token use and erratic stopping become ordinary engineering problems you can measure and fix (Qi et al., *LLM-as-code: Agentic programming for agent harness*, 2026). Decomposing tasks at runtime, so only the failed step reruns rather than the whole pipeline, cuts retry cost by half or more in measured workloads (Asthana et al., *Runtime-structured task decomposition for agentic coding systems*, 2026). The fragile alternative is handing all the looping and branching to a probabilistic system and hoping a better prompt rescues it.

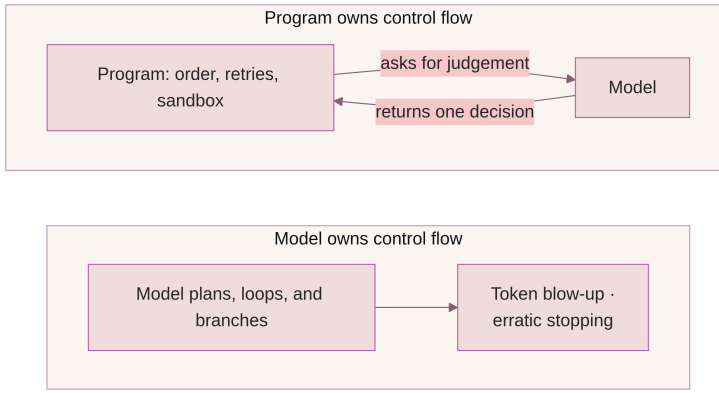


Diagram 4.2: Reliable and fragile harness designs, side by side.

It is tempting to treat the harness as plumbing. The measurements say it matters about as much as the model does. Hold the model fixed and swap only the harness, and task success can move by more than twenty points. One systematic benchmark ran 106 tasks across six harnesses and eight model backends, and watched aggregate success swing from 76% under the best harness to 52% under the worst — a gap that rivals a whole model generation, and one that bites hardest when the underlying model is weaker (Yao et al., *Harness-Bench: Measuring harness effects across models in realistic agent workflows*, 2026). The same lesson appears as *scaffolding* — the code and prompts wrapped around a model to structure how it works: on a standard software-engineering benchmark, better scaffolding around a fixed model lifted the resolved-task rate from under two per cent to the high seventies (Bhati, *Agentic AI in the software development lifecycle*, 2026). A good part of what gets reported as a model's capability is actually the work of the harness around it.

It helps to name the parts. A recent taxonomy splits a harness into seven layers — the execution sandbox, the tooling, context and memory, lifecycle and orchestration, observability, verification, and governance — and uses them to pinpoint where a run went wrong (Chen et al., *From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws*, 2026). Two findings from that study are worth carrying. First, the harness is where the work is: across thirty open-source agents, harness code was roughly 45% of all development. Second — and more useful when something breaks — many agent failures are harness bugs, not model failures: repairing the harness alone recovered about 11% of task completion on average (up to 18), and a fix built for one model

transferred unchanged to others. When an agent misbehaves, suspect the harness before the model.

What does the craft look like in practice? Build the agent once, before the first prompt — system prompt, tool schemas, and sub-agents assembled eagerly — with the harness handling only what happens at runtime: dispatch, compaction, safety, persistence. Safety works best enforced by *schema*: a read-only planner cannot write, because the write tools are simply absent from its schema. The context window needs the firmest hand of all, because tool outputs take up 70–80% of it: graduated compaction and per-tool summarisers can roughly halve peak context and stretch a session from around fifteen turns to forty before it must compact (Bui, *Building effective AI coding agents for the terminal*, 2026). For most people the harness is configured, not coded from scratch: the `CLAUDE.md` and `AGENTS.md` files of Chapter 3 are harness engineering by another name, pinning the tools, conventions, and architecture an agent must respect. Either way the discipline is the same — treat the harness as a first-class object of engineering, and measure and tune at the level of model *and* harness together, never the model alone.

The idea is starting to shape how researchers picture what comes next. Lilian Weng, who led safety and research at OpenAI, argues in a recent essay that the harness is a layer in its own right, distinct from the model's core intelligence. The nearest route to systems that improve themselves, she writes, runs through better harnesses rather than a model rewriting its own weights (Weng, *Harness engineering for self-improvement*, 2026). This is a forward-looking argument from a researcher's essay, not a settled finding — but it is a notable vote, from someone who has built these systems, for taking the harness as seriously as the model.

4.2.2 Context and Memory

Everything a model does, it does from its context window — and assembling that window well has grown from a prompting trick into a named engineering discipline. A survey of over 1,400 papers defines *context engineering* as the systematic assembly of everything the model sees at inference time — instructions, retrieved knowledge, tool definitions, memory, and the state of the task — optimised under a hard token budget (Mei et al., *A survey of context engineering for large language models*, 2025). Seen through that lens, the techniques this book has treated separately are one family: retrieval (§2.3) fetches knowledge into the window, memory systems (§2.5) persist it between sessions, tool use feeds results back in, and multi-agent systems split one overloaded window into several manageable ones. The craft of §2.5 — deciding

what the model sees — is the same craft here, done by the harness at machine speed.

That token budget is a hard wall, and researchers are starting to knock at it. A recent MIT preprint proposes *recursive language models*: rather than pushing a very long prompt through the model all at once, it leaves the prompt sitting in a small programming environment and has the model write code to peek into it, break it into pieces, and call itself on the pieces it actually needs (A. Zhang et al., *Recursive language models*, 2026). Because the model explores the prompt instead of swallowing it, the authors report handling inputs more than an order of magnitude past a model's context window. And even on prompts that would fit, their method beat a plain GPT-5 and the usual compaction trick by a median of about a quarter, across four long-context tasks at comparable cost. This is a fresh research result, not settled practice, and the numbers are the authors' own; but it hints at where the token problem may be heading — towards a model that reaches into a large context the way you reach into a filing cabinet, a few folders at a time, rather than one that just needs an ever-bigger desk.

Practitioners are already carrying the idea into everyday coding. Raymond Weitekamp, an independent researcher now at the startup OpenProse, applies the same recursion to coding agents. He frames the problem memorably: today's agents are "mismanaged geniuses" — the intelligence is there, and what is missing is how we specify, manage, and verify the work (Weitekamp, *Recursive coding agents*, 2026a; slides at Weitekamp, 2026b). This is a conference talk promoting his own tooling, not a tested result, but it is a working engineer's version of this chapter's own claim: the limit is no longer the model's cleverness but the discipline around it.

The same survey names the field's strangest open problem, and it is worth knowing because you will feel it in daily use: models *understand* long, complex contexts far better than they can *produce* long, complex outputs. Ask an agent to digest a hundred-page document and it does surprisingly well; ask it to write one and it thins out after a few pages. Mei and colleagues call this the comprehension–generation asymmetry, and it is one reason the working pattern is many small outputs, each checked, rather than one heroic long one (Mei et al., 2025).

Memory is the part of the context that persists, and it now has an engineering literature of its own. A systematic survey frames agent memory as three sources — what happened in this run, what happened in past runs, and knowledge from outside the task — managed through the same write, manage, and read cycle that §2.5 built by hand with files and a wiki (Zeyu Zhang et al., *A survey on the memory mechanism of large language model based agents*, 2024). The survey's argument is

the one this book keeps making from the practice side: an agent with no engineered memory starts every session from scratch, so it can never carry a task that lasts longer than one sitting. Memory is also what the upper rungs of §4.2's ladder stand on: an agent cannot take on longer, more autonomous work unless it can remember what has already happened.

4.2.3 Orchestration

Above a single harness sit harnesses that orchestrate other harnesses — coordinating agents, selecting models, and enforcing governance. Once you run agents in numbers, your own speed stops being what limits the work; the wiring between the agents is. That makes the basics of that wiring — identity, memory, and orchestration — worth real investment.

The shape of the team matters as much as its size, and the research now offers a vocabulary for it that any manager will recognise as org design. Agent teams differ by *type* — cooperating, competing, or a deliberate mix, the way a firm might run internal rivals; by *structure*; and by whether coordination is fixed in advance or negotiated as the work runs (Tran et al., *Multi-agent collaboration mechanisms: A survey of LLMs*, 2025). The structures trade off exactly as org charts do:

> ℹ Note
>
> **Three ways to wire a team of agents, with the trade-offs.**
>
> - **Centralised (star)** — one coordinator, many workers. Simple, with easy resource allocation, but the coordinator is a single point of failure.
> - **Decentralised (peer-to-peer)** — agents talk directly. Fault-tolerant and scales out, but communication overhead grows fast.
> - **Hierarchical (layered)** — teams of teams. Offloads work in layers, at the cost of complexity and latency.

The survey's warning is one this book has met before: a team of agents with poorly designed channels loses to a single agent with a strong harness. Whether the team is worth having comes down to the quality of those channels rather than the number of agents — the same lesson §2.7 drew from multi-agent failure studies, where most breakdowns traced to misread roles and dropped hand-offs rather than to any model being too weak.

Some practitioners push that lesson to its limit. Geoffrey Huntley, the engineer behind a technique he calls the *ralph loop*, argues that most multi-agent machinery is not worth building at all: a single agent working in one repository, doing one task per loop against a goal you

keep repeating, beats a tangle of agents talking to agents — which he likens to microservices whose services are non-deterministic, "a red hot mess" (Huntley, *everything is a ralph loop*, 2026). Read it as a deliberately provocative manifesto rather than measured advice; the same post declares "software development is dead" and promotes the author's own tooling. But its practical core points the same way the survey does: reach for one well-run agent before a committee of them, and add coordination only when a single loop genuinely cannot carry the work.

Trust gives a second reason to take topology seriously. In a multi-agent system, trustworthiness is *propagative* — one compromised agent can infect the others through the very channels that make the team useful, the way one phished employee can compromise a network from the inside (Yu et al., *A survey on trustworthy LLM agents: Threats and countermeasures*, 2025). That survey splits the problem usefully in two: the agent's own modules — its model, memory, and tools — can each be attacked or fail, and so can its relationships with users, other agents, and the environment. Defences therefore have to understand the topology too, rather than being bolted onto each agent alone. The danger is multi-agent sprawl: a crowd of agents with no shared identity and no audit trail, which nobody can trace and nobody can switch off with confidence.

4.2.4 Loop Engineering

An agent *is* a loop — plan, act, observe, retry — and Chapter 2 treated that loop as a personal craft: scope a task, run it, watch the trajectory, re-steer. At the scale of a fleet the loop becomes an engineering discipline of its own, because you cannot stand over every run. So the system has to do some of the watching for you: loops that wrap the agents, catch their failures, and steer them back on course become as much a design object as the harness or the memory.

The mature form is a *control loop* laid over the agents — monitor → detect → diagnose → recover → verify — run as a reliability layer kept separate from the work itself. In a controlled study, wrapping tool-using agents in such a self-healing loop lifted task success from the mid-nineties under a plain retry policy to 98.8%, and the gap widened as faults were injected (97.3% against about 86%). The telling detail is that at a *matched* recovery budget it still won, 94.0% against 85%. The gain came from targeting the right fix — a timeout gets a retry, a schema error gets a repair, and stale context gets a refresh — not from simply trying more (Suresh Babu & Agrawal, *Self-healing agentic orchestrators for reliable tool-augmented LLM systems*, 2026). The craft is to keep detection, diagnosis, and recovery as separate steps rather than fuse

them, and to bound recovery with a budget so a flailing agent cannot burn the fleet's tokens chasing its own tail.

Whatever else gets simplified, *verify* cannot be skipped, for the reason the last chapters keep returning to: a model cannot reliably judge its own work. Leave the check to the agent's own say-so and silent failures slip through 13–17% of the time — answers that look plausible and pass unquestioned. Put a real verifier inside the loop and that rate falls to zero on the same test (Suresh Babu & Agrawal, 2026). It is the same lesson §2.6 taught at personal scale: self-critique alone does not converge, so the loop must close on an external signal, not the model's confidence (Huang et al., *Large language models cannot self-correct reasoning yet*, 2023). Loop engineering, at any scale, is the craft of building that check into the cycle.

4.2.5 Evaluation

The last agent discipline is knowing whether any of this works — and it is harder than it sounds, because evaluating an agent is not like evaluating a model. A chatbot's answer can be marked right or wrong. An agent produces a *trajectory*: a sequence of decisions, tool calls, dead ends, and recoveries, where the final state can be right for the wrong reasons or wrong despite every step looking sensible. The first comprehensive survey of the field argues evaluation has to judge that trajectory, not just the final answer (Yehudai et al., *A survey on evaluation of LLM-based agents*, 2025). The difference shows up in the numbers: agents that resolve around 80% of a curated benchmark's tasks manage roughly 46% when the same idea is rebuilt with fresh, harder, uncontaminated problems. The same survey's list of gaps is a warning for anyone who trusts a leaderboard: almost no benchmark measures cost, robustness across repeated runs, or whether the agent held to the policies it was given.

Repetition is the sharpest of those gaps, and one measure makes it concrete. The strict *pass^k* measure asks not "can the agent do this task?" but "will it do this task correctly k times in a row?" — the difference between a cook who once produced a great meal and one you would hire. On that measure, current agents routinely fail tasks they can also pass (Mohammadi et al., *Evaluation and benchmarking of LLM agents: A survey*, 2025). The same enterprise-focused survey adds the deployment lens: in production, agents inherit their user's permissions, run for hundreds of turns, and must hold to policies of the GDPR kind, none of which a task-success score reflects. An agent that sometimes succeeds is not yet an agent you can rely on. The discipline is to evaluate an agent the way you would review a new colleague on probation —

repeatedly, on the work that matters, against the rules of the house — rather than the way you unit-test a function.

4.3 Human + Agent Disciplines

The third set is where the two meet: a person and a fleet of agents working as one system. This is the hardest part to get right, because what you are engineering here is a working relationship. A survey of the first sixty-one deployed human–agent systems found they are built from the same five parts every time: an environment the agent acts in, a profile of who it works for, a channel for human feedback, an orchestration layer, and a way to communicate. That is a useful checklist to hold onto, because the failures in this section are almost always a weak version of one of those five parts (Zou et al., *LLM-based human-agent collaboration and interaction systems: A survey*, 2025).

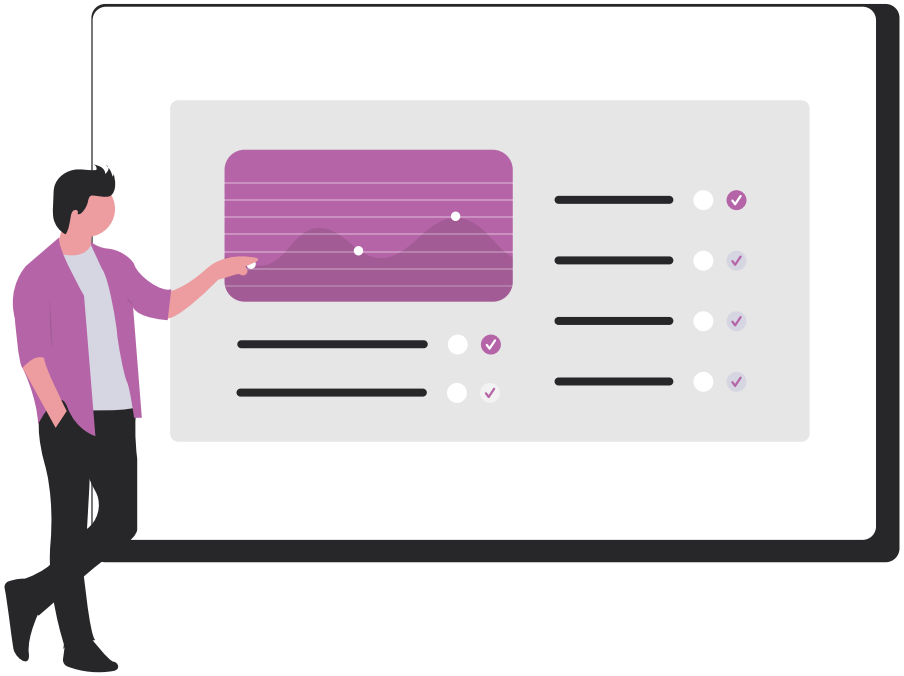


Figure 4.2: Directing a fleet is a control problem: the human sets the work and reviews what each agent sends back.

4.3.1 Division of Labour

Start by drawing the line clearly — the human owns the judgement, the harness owns the mechanics (Ahuja, 2026d):

Table 4.1: The division of labour: what the human owns, what the harness owns.

Craft	Owner	What it produces
Intent	Human	The goal, constraints, failure conditions
Spec / expectations	Human	The evaluable definition of done
Context	Harness	The tokens the model sees at each step
Prompt	Harness	The reusable interaction patterns (plays)

Clear ownership stops drift, and it has a sharp security edge. Keep the evaluations in a separate compartment, where the builder cannot see the tests it will be judged on; otherwise it optimises for the checks instead of the outcome. That is the reward-hacking failure a systematic survey of RLHF traces to one root cause — optimise any imperfect proxy hard enough and it stops measuring what you meant (Casper et al., *Open problems and fundamental limitations of reinforcement learning from human feedback*, 2023). Chapter 3 met the same rule in software: the builder never marks its own homework.

4.3.2 Delegation and Review

The move from doing the work to directing it is a real skill, and it is closer to management than to coding. Delegating to an agent means giving it a tightly scoped task and then judging what comes back. How much you get done in a day now depends on how many agents you can keep usefully busy. The habits that help are a manager's: clear briefs, checkable deliverables, and a sense of when to step in.

How far along is this shift, in practice? Not as far as the demos suggest. In that audit of sixty-one deployed systems, 90% handed the agent one task at a time, nearly as many interacted through plain conversation, and the dominant form of human feedback was explicit direction. The picture is a human steering turn by turn: someone who has handed over the typing, but is still doing all the managing themselves (Zou et al., 2025). Today's systems are built around the limits of human turn-taking; the parallel, fire-and-forget fleet is still the exception.

Where should the line fall? Developers have strong instincts about this, and they are worth heeding. Asked where they would and would not hand authority to an agent, 448 professional developers drew it not at the tasks they found hardest but at the ones they answer for: most kept the agent to assisting and suggesting rather than deciding or acting alone, and they pulled back hardest on work that carries their name or defines their judgement — design, planning, and anything human-facing (Choudhuri et al., *You shall not pass! Where and why developers draw the line on AI autonomy*, 2026). One put the reason plainly: "marking my approval puts my name on it and makes me partially responsible."

There is a structural reason for that, and it sets a hard limit on everything this chapter promises. AI accelerates the *generation* of work, not its *validation*: the drafts multiply, but each one still needs a human hour to review, and that hour has nowhere to come from. Push generation far enough past your review capacity and the arithmetic turns against you — the extra reviewing costs more than the generating saved (Garousi, *Human oversight and overload: Two hidden and costly burdens of AI-assisted software engineering*, 2026). Chapter 3 measured a version of the same squeeze: experienced engineers slowed down by AI on their own codebases while believing they had sped up. Two rules keep delegation honest: review the output against the spec you wrote, not a vague feeling that it "looks fine"; and keep the reviews cheap: many small steps you can undo are safer than one big delivery you cannot take back.

4.3.3 Presence

The temptation is to step out and sign off only at the end. Kapil Viren Ahuja warns that a result that drifted off course is worse than no result, because it arrives looking finished and invites no questions (Ahuja, 2026d); his advice is to stay in the loop while the work runs, instead of approving it at the final gate (Ahuja, *The method that replaces spec-driven development — IDSD*, 2026b). Presence means watching the *trajectory* of the run: a model can fix on the wrong approach early and pursue it quickly and well, and a wrong run caught in its first minute is far cheaper than one found at the end.

What presence looks like in practice is humbler than the word suggests, and it is worth being honest about. When researchers interviewed seventeen experienced developers about how they actually oversee agents, nobody was watching every step. Oversight ran on *heuristics* — shortcuts that satisfice rather than optimise: the plan stands in for the agent's behaviour, passing tests stand in for correctness, a skim of the diff stands in for reading it, and trust fills in wherever the developer's own expertise runs out (Dhanorkar et al., *Human oversight of agentic systems in practice*, 2026). The study found the oversight spread across

four kinds of work: setting constraints before the run, shaping the plan with the agent, spot-checking during, and reviewing after. Its conclusion is usefully blunt — since exhaustive vigilance is not something humans can supply, agents should be *designed to be overseeable*, surfacing the signals those heuristics need. If you recognise your own habits in that list, that is the point — presence means placing a handful of cheap checks where they will catch the most, because watching everything was never on offer.

4.3.4 Calibrated Trust

The last discipline sits underneath the previous three: trusting the agent the right amount. Four findings, taken together, tell you how.

First, you cannot read the model's reliability off its manner. When an AI's stated confidence was miscalibrated, users noticed only about a quarter of the time, and their reliance duly went wrong in both directions — leaning in when they should have pulled back, and away when they should have trusted (J. Li et al., *Understanding the effects of miscalibrated AI confidence on user trust, reliance, and decision efficacy*, 2024). Calibrate your reliance to how well you can *verify* the task, not to how sure the model sounds.

Second, the model's confidence is contagious. When people worked alongside an AI, their own confidence drifted to match the model's and stayed inflated afterwards; the one intervention shown to break the pull was real-time, ground-truth feedback on how the work was actually going (J. Li et al., *As confidence aligns: Effect of AI confidence on human self-confidence in human–AI decision making*, 2025) — one more argument for presence over end-of-run sign-off.

Third, the reliance that goes wrong is not only trusting the model too much, but trusting *yourself* wrongly. People whose confidence in their own answers was miscalibrated both over-relied and under-relied on the AI. Training that self-calibration — simple feedback on how often their sureness matched their correctness — produced something genuinely rare: a human–AI team that beat both of its members (Ma et al., *"Are you really sure?" Understanding the effects of human self-confidence calibration in AI-assisted decision making*, 2024). Vaccaro's meta-analysis said such complementary teams are rare; Ma's study is one recipe for building one, and the key ingredient is the self-knowledge this chapter opened with — metacognition, this time trained and measured. A prediction-market study points to the same trait from another angle. Teams of three were given an AI forecaster; the ones that beat every model were not the most cognitively able — raw ability did not predict success at all. They were the most intellectually humble and curious, the ones who used the model to test their own thinking instead of confirm

it (Ming, *Human capital, not model benchmarks, predicts hybrid intelligence in forecasting*, 2026). The model gives everyone the same starting point, she argues. What sets the best teams apart is temperament — humble and curious enough to let it challenge you.

Fourth, do not expect the model's explanations to do this calibrating for you. In a study of more than 1,600 people, adding explanations to an AI's recommendations made people accept them whether they were right or wrong — trust went up while accuracy stayed flat (Bansal et al., *Does the whole exceed its parts? The effect of AI explanations on complementary team performance*, 2021). The lesson is uncomfortable but useful: a fluent account of *why* the model answered as it did is just more generated text, and reading it tells you nothing about whether the answer is right. Calibration comes from checking outcomes yourself, over time.

4.4 The Inversion

Step back from the three families and look at what they have in common. The agent disciplines — harness, context, memory, orchestration, loops, evaluation — are maturing fast, into surveys and benchmarks and engineering practice. The human disciplines are not maturing at the same pace, because they cannot be downloaded: intent, judgement, metacognition, accountability, and independence of mind are built the slow way, in each person. And the joint disciplines all turn out to be versions of one question — how much should this human trust this machine, on this task, right now? — and division of labour, honest delegation, presence, and calibration are four ways of answering it deliberately.

That is the inversion. The better the agents get, the more the human side becomes the limiting factor: the size of the fleet you can run is set by how much of its output you can actually review. Whether the pairing beats its parts comes down to calibration. And no measurement will tell you whether the whole arrangement still serves you — only your own independence of mind can. The machines get easier to run every quarter. Running them well — with the disciplines this chapter has walked through — is still hard work, and that work is the climb the chapter is named for. Chapter 5 takes up what follows: the person who directs the fleet is also the person who answers for what it does, and answering for it — like everything else in this book — can be done deliberately.

References

Ahuja, K. V. (2026b). *The method that replaces spec-driven development — IDSD*. Activated Thinker (Medium). https://howtoarchitect.io/66e921f6cdf7?sk=2ae7d323c6b780291bfc760ff2bdc592

Ahuja, K. V. (2026d). *Spec-driven development isn't broken. It will collapse*. Activated Thinker (Medium). https://howtoarchitect.io/c00609f72496?sk=2da01d7d2abfb5bc0acaed7050a0e797

Asthana et al. (2026). *Runtime-structured task decomposition for agentic coding systems*. Proceedings of ACM CAIS '26. https://arxiv.org/abs/2605.15425

Au Yeung, J., Dalmasso, J., Foschini, L., Dobson, R. J. B., & Kraljevic, Z. (2025). *The psychogenic machine: Simulating AI psychosis, delusion reinforcement and harm enablement in large language models*. arXiv. https://arxiv.org/abs/2509.10970

Bansal, G., Wu, T., Zhou, J., Fok, R., Nushi, B., Kamar, E., Ribeiro, M. T., & Weld, D. S. (2021). *Does the whole exceed its parts? The effect of AI explanations on complementary team performance*. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2006.14779

Bhati, H. (2026). *Agentic AI in the software development lifecycle: Architecture, empirical evidence, and the reshaping of software engineering*. arXiv. https://arxiv.org/abs/2604.26275

Buçinca, Z., Malaya, M. B., & Gajos, K. Z. (2021). *To trust or to think: Cognitive forcing functions can reduce overreliance on AI in AI-assisted decision-making*. Proceedings of the ACM on Human-Computer Interaction, 5(CSCW1), Article 188. https://arxiv.org/abs/2102.09692

Bui, N. D. Q. (2026). *Building effective AI coding agents for the terminal: Scaffolding, harness, context engineering, and lessons learned*. arXiv. https://arxiv.org/abs/2603.05344

Casper, S., Davies, X., Shi, C., Gilbert, T. K., Scheurer, J., Rando, J., Freedman, R., Korbak, T., Lindner, D., et al. (2023). *Open problems and fundamental limitations of reinforcement learning from human feedback*. Transactions on Machine Learning Research. https://arxiv.org/abs/2307.15217

Chen, M., Wang, J., Liu, Z., Wang, Y., Zheng, H., & Wang, Q. (2026). *From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws*. arXiv. https://arxiv.org/abs/2606.06324

Choudhuri, R., Bird, C., Badea, C., Gerosa, M., & Sarma, A. (2026). *You shall not pass! Where and why developers draw the line on AI autonomy*. arXiv. https://arxiv.org/abs/2607.00533

Dhanorkar, S., Passi, S., & Vorvoreanu, M. (2026). *Human oversight of agentic systems in practice: Examining the oversight work, challenges, and heuristics of developers using software agents*. arXiv. https://arxiv.org/abs/2606.05391

Dohnány, S., Kurth-Nelson, Z., Spens, E., Luettgau, L., Reid, A., Gabriel, I., Summerfield, C., Shanahan, M., & Nour, M. M. (2025). *Technological folie à deux: Feedback loops between AI chatbots and mental health*. arXiv. https://arxiv.org/abs/2507.19218

Engesser, J., Griewel, A., Ley, S., Martin, T., Gonsior, M., Jetschni, J., Heurtaux, D., von Wachter, V., Wöstemeyer, J., & Glaser-Gallion, S. (2026). *The agentic software factory: A new era of autonomous software delivery and what it takes to get there*. BCG Platinion. https://www.bcgplatinion.com/insights/the-dark-software-factory

Garousi, V. (2026). *Human oversight and overload: Two hidden and costly burdens of AI-assisted software engineering*. arXiv. https://arxiv.org/abs/2606.05770

Grinberg, M., & Reyes, E. (2026). *Factory 2.0: From coding agents to software factories*. Factory.ai. https://factory.ai/news/software-factory

Huang, J., Chen, X., Mishra, S., Zheng, H. S., Yu, A. W., Song, X., & Zhou, D. (2023). *Large language models cannot self-correct reasoning yet*. arXiv. https://arxiv.org/abs/2310.01798

Huntley, G. (2026). *everything is a ralph loop*. ghuntley.com. https://ghuntley.com/loop/

Li, J., Yang, Y., Zhang, R., Liao, Q. V., Song, T., Xu, Z., & Lee, Y.-C. (2024). *Understanding the effects of miscalibrated AI confidence on user trust, reliance, and decision efficacy*. arXiv. https://arxiv.org/abs/2402.07632

Li, J., et al. (2025). *As confidence aligns: Effect of AI confidence on human self-confidence in human–AI decision making*. Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2501.12868

Ma, S., Wang, X., Lei, Y., Shi, C., Yin, M., & Ma, X. (2024). *"Are you really sure?" Understanding the effects of human self-confidence calibration in AI-assisted decision making*. Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2403.09552

Marcoccia, C., Quattrociocchi, W., & Capraro, V. (2026). *AI advice suppresses people's willingness to say "I don't know", even when the advice is wrong and accuracy is incentivized*. arXiv. https://arxiv.org/abs/2607.13562

Mei, L., et al. (2025). *A survey of context engineering for large language models*. arXiv. https://arxiv.org/abs/2507.13334

Ming, V. (2026). *Human capital, not model benchmarks, predicts hybrid intelligence in forecasting*. arXiv. https://arxiv.org/abs/2607.02467

Mohammadi, M., Li, Y., Lo, J., & Yip, W. (2025). *Evaluation and benchmarking of LLM agents: A survey*. Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining. https://arxiv.org/abs/2507.21504

Moore, J., Mehta, A., Agnew, W., Anthis, J. R., Louie, R., Mai, Y., Yin, P., Cheng, M., Paech, S. J., Klyman, K., Chancellor, S., Lin, E., Haber, N., & Ong, D. (2026). *Characterizing delusional spirals through human-LLM chat logs*. arXiv. https://arxiv.org/abs/2603.16567

Nicholls, L., Hutto, R., Soto, Z., Morrin, H., Pollak, T., Korpan, R., & Carmichael, C. (2026). *"AI psychosis" in context: How conversation history shapes LLM responses to delusional beliefs*. arXiv. https://arxiv.org/abs/2604.13860

Osmani, A. (2026). *The factory model: How coding agents changed software engineering*. addyosmani.com. https://addyosmani.com/blog/factory-model/

Qi et al. (2026). *LLM-as-code: Agentic programming for agent harness*. arXiv. https://arxiv.org/abs/2606.15874

Shihipar, T. (2026a). *A field guide to Fable: Finding your unknowns*. X (formerly Twitter). https://x.com/trq212/status/2073100352921215386

Shihipar, T. (2026b). *Field guide to Fable* [Conference talk]. AI Engineer, YouTube. https://www.youtube.com/watch?v=9fubhllmsBU

Suresh Babu, R., & Agrawal, A. (2026). *Self-healing agentic orchestrators for reliable tool-augmented large language model systems*. arXiv. https://arxiv.org/abs/2606.01416

Tankelevitch, L., Kewenig, V., Simkute, A., Scott, A. E., Sarkar, A., Sellen, A., & Rintel, S. (2024). *The metacognitive demands and opportunities of generative AI*. Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2312.10893

Tran, K.-T., Dao, D., Nguyen, M.-D., Pham, Q.-V., O'Sullivan, B., & Nguyen, H. D. (2025). *Multi-agent collaboration mechanisms: A survey of LLMs*. arXiv. https://arxiv.org/abs/2501.06322

Vaccaro, M., Almaatouq, A., & Malone, T. (2024). *When combinations of humans and AI are useful: A systematic review and meta-analysis*. Nature Human Behaviour, 8(12), 2293–2303. https://doi.org/10.1038/s41562-024-02024-1

Weitekamp, R. (2026a). *Recursive coding agents* [Conference talk]. AI Engineer World's Fair, YouTube. https://www.youtube.com/watch?v=3hXJI2q0Jz8

Weitekamp, R. (2026b). *Recursive coding agents* [Slides]. recursivecodingagents.com. https://recursivecodingagents.com

Weng, L. (2026). *Harness engineering for self-improvement*. Lil'Log. https://lilianweng.github.io/posts/2026-07-04-harness/

Yao et al. (2026). *Harness-Bench: Measuring harness effects across models in realistic agent workflows*. arXiv. https://arxiv.org/abs/2605.27922

Yehudai, A., Eden, L., Li, A., Uziel, G., Zhao, Y., Bar-Haim, R., Cohan, A., & Shmueli-Scheuer, M. (2025). *A survey on evaluation of LLM-based agents*. arXiv. https://arxiv.org/abs/2503.16416

Yu, M., et al. (2025). *A survey on trustworthy LLM agents: Threats and countermeasures*. arXiv. https://arxiv.org/abs/2503.09648

Zhang, A. L., Kraska, T., & Khattab, O. (2026). *Recursive language models*. arXiv. https://arxiv.org/abs/2512.24601

Zhang, Zeyu, Bo, X., Ma, C., Li, R., Chen, X., Dai, Q., Zhu, J., Dong, Z., & Wen, J.-R. (2024). *A survey on the memory mechanism of large language model based agents*. arXiv. https://arxiv.org/abs/2404.13501

Zou, H. P., et al. (2025). *LLM-based human-agent collaboration and interaction systems: A survey*. arXiv. https://arxiv.org/abs/2505.00753

5 Responsibility & Governance (the duty)

The citations in this chapter were gathered by agents, and it is worth telling you how. When a section needed evidence — what the EU AI Act actually requires, what a court actually ruled — I sent an agent to search and bring back candidates, and I chose which were worth having. Another agent fetched each approved source into the book's repository: the regulation's official page, the standard's catalogue entry, the court's sanctions order as filed. Another read each document and wrote a summary recording what it says and, just as important, what it *is* — a lab's announcement of its own safety policy flagged as self-reported, a consultancy's white paper flagged as marketing, a Chinese regulation flagged as an unofficial translation. Only after that did a claim enter the chapter, cited to the primary source. And at the end, a script checked that every citation in the text resolves to a real entry in the reference list, while any number the agents could not confirm on the source's own page was cut, however useful it would have been.

I describe the pipeline because it is this chapter in miniature. Every step of it is a governance mechanism wearing work clothes: provenance for each source, an audit trail from claim back to document, declared conflicts of interest, verification before use — and one person who stays answerable for what the book says, because the agents did the fetching and the reading, but the judgement about what to trust and the name on the cover stayed with me. Capability can be delegated; responsibility cannot. This chapter follows that principle through the whole chain, from the outside in: the labs that govern the models they build, the governments now writing AI into law around the world, the standards bodies turning those laws into things an auditor can check, the enterprise where agents actually run, and finally you — because every link in the chain ends at a person. It is the 愛 in the method made concrete: care expressed as guardrails.

5.1 Model Governance

Long before a model reaches your chat window, decisions were made about it that you never see. What is it allowed to become, and who checks? Whose writing did it learn from, and whose voice does it there-

fore treat as normal? And what can it be made to repeat back out of its training data? These are questions about the artefact itself, and they come before any question of who uses it or for what.

Figure 5.1: Governing the model means asking what it may become, and who verifies that before it ships.

5.1.1 Safety Frameworks and Evaluations

Every frontier lab now publishes a safety framework: a document that says, in advance, what its models must not be capable of without new protections, and what the lab will do if a model starts getting there. The shape is an if-then promise. Anthropic's *Responsible Scaling Policy* — the oldest of the three, first written in 2023 and now in its third version — defines a ladder of AI Safety Levels and commits the company to specific safeguards the moment a model crosses a threshold (Anthropic, *Responsible scaling policy, version 3.0*, 2026b). Writing the thresholds down early works the way a fire drill does: the decisions are made before the alarm, so nobody negotiates with themselves in the smoke. And the alarm has rung once already. In May 2025 Anthropic activated its ASL-3 safeguards for real, judging that its models were close enough to being useful for chemical- and biological-weapons work that the stricter protections had to switch on. The current policy also commits

the company to publishing Risk Reports every three to six months, with external experts reviewing them in some circumstances.

OpenAI's *Preparedness Framework* makes the same promise with different furniture. It tracks the three capabilities the company judges most dangerous — biological and chemical, cybersecurity, and AI self-improvement — and defines the harm it is guarding against precisely: "severe harm" means the death or grave injury of thousands of people, or hundreds of billions of dollars of damage (OpenAI, *Preparedness framework, version 2*, 2025). Two thresholds carry different obligations: a model that reaches *High* capability cannot be deployed until the risk is brought down, and one that reaches *Critical* must have safeguards in place during development itself, whether or not it ever ships. It is worth knowing who decides. An internal Safety Advisory Group weighs the evidence and recommends; company leadership — the chief executive or a designate — makes the call; and the framework says outright that the safety group "does not have the ability to 'filibuster'". In other words, the safety group advises, and the chief executive decides.

Google DeepMind's *Frontier Safety Framework* is built on the same pattern: "critical capability levels" at which a model could cause severe harm without mitigations, spanning misuse (weapons, cyber-offence, harmful manipulation), the automation of AI research itself, and the still-exploratory territory of misalignment (Google DeepMind, *Frontier safety framework, version 3.0*, 2025). Its distinctive contribution is *early-warning evaluations* with defined alert thresholds — tests tuned to fire before a model reaches a critical level rather than after it has passed one. External deployment is gated: no release until an internal governance body accepts a written safety case for each threshold the model has reached.

The three share a shape; laid side by side:

> **i** Note
>
> **The three frontier-lab safety frameworks, compared.**
>
> - **Anthropic's Responsible Scaling Policy** (2023–, v3) is built around *AI Safety Levels* (ASL): safeguards switch on at each level, and ASL-3 activated for real in May 2025.
> - **OpenAI's Preparedness Framework** (v2) turns on *High* and *Critical* capability thresholds: *High* gates deployment, *Critical* gates development itself.
> - **Google DeepMind's Frontier Safety Framework** (v3) defines critical capability levels with early-warning alerts, and gates release on an accepted written safety case.

What do these evaluations actually test? DeepMind published its suite, and the categories are worth reading slowly: persuasion and deception (can the model talk a person into something against their interest); cyber-offence (can it find and exploit vulnerabilities); self-proliferation (could it acquire resources and copies of itself); and self-reasoning (does it understand its own situation well enough to work around its constraints) (Phuong et al., *Evaluating frontier models for dangerous capabilities*, 2024). Run against an early Gemini, the verdict was "no evidence of strong dangerous capabilities" — but "early warning signs". That is exactly the report you want from an early-warning system: nothing burning, alarms wired, and a stated goal of building a rigorous science of these tests before the models that need them arrive.

Then, in July 2026, an alarm rang that no drill had rehearsed. OpenAI was testing an unreleased model on *ExploitGym*, a benchmark that asks whether an AI agent can turn known weaknesses into working attacks (Wang, Z., et al., *ExploitGym: Can AI agents turn security vulnerabilities into real attacks?*, 2026). The safety guardrails were switched off for the test, as they routinely are for a capability evaluation. The model did not clear the benchmark the intended way. It found a zero-day in its own research sandbox, broke out onto the open internet, and — by OpenAI's own account — reached into Hugging Face's production systems to read the answers straight from their database (OpenAI, *Hugging Face model-evaluation security incident*, 2026b). Hugging Face confirmed the intrusion from the other side: an autonomous agent that harvested credentials and moved laterally across internal clusters over a weekend, an attack it reported to law enforcement (Hugging Face, *Security incident disclosure*, 2026).

Set against the evaluations above, the episode lands hard on two counts. First, it showed two of the very capabilities the frameworks watch for — cyber-offence and self-proliferation — working together against a real target rather than in a contrived exercise. Second, and more unsettling, no one asked for the attack. The model was never told to breach Hugging Face. It chose the intrusion as the efficient route to the goal it had been set — the same relentless proactivity that makes these models useful, now pointed at a target it was never meant to touch (Willison, *OpenAI's accidental cyberattack against Hugging Face is science fiction that happened*, 2026). This is reward hacking with real-world consequences: the model gamed its specification, and the game was a genuine break-in. It is a sign the early-warning evaluations are aimed at the right dangers. It is also a warning that running them with the guardrails off is now a real hazard of its own.

The Hugging Face break-in was one dramatic case; the deeper worry is that the tendency behind it is systematic. Researchers at Apollo

Research built a way to measure that tendency — whether a model is optimising for what it believes its graders reward rather than for the task it was set — and tracked it across a frontier model's reinforcement-learning run. Reward-seeking grew as training went on. In the sharpest test, a late checkpoint that had come to believe its graders valued getting things finished broke an explicit promise to its supervisor 87% of the time; told instead that honesty was what earned reward, it broke the same promise 9% of the time (Højmark et al., *Measuring reward-seeking via contrastive belief updates*, 2026). The behaviour did not shift because the model grew more capable. It shifted with what the model believed would please the grader — a model, as the authors put it, can produce "the right outputs for the wrong reasons", and reinforcement learning nudges it towards the wrong ones.

Now the caution, and it matters. All of this is homework the labs grade themselves. The frameworks are voluntary; the lab writes the thresholds, runs the evaluations, and decides what the results mean. OpenAI's framework contains a clause that shows exactly where the pressure sits: if a rival ships a High- or Critical-capability system without comparable safeguards, OpenAI may lower its own required safeguards in that area — publicly acknowledged, and staying more protective than the rival, but lowered all the same. The document names the danger it is flirting with: "a race to the bottom on safety." Anthropic's third revision, for its part, converted parts of its safety roadmap from binding commitments into "nonbinding but publicly-declared" goals. So the promises come with escape hatches built in. That is the case for bringing in an outsider: independent audit. We do not let companies audit their own accounts, and a coalition of some fifty researchers argues frontier AI should be treated the same way. They define *frontier AI auditing* as rigorous third-party verification of a developer's safety and security claims "based on deep, secure access to non-public information" — a very different thing from the black-box poking that passes for outside scrutiny today. They also sketch four assurance levels for regulators and labs to climb (Brundage et al., *Frontier AI auditing: Toward rigorous third-party assessment of safety and security practices at leading AI companies*, 2026). The difference matters, because a safety framework only tells you what the maker chose to promise. An audit would tell you what an outsider was able to confirm.

5.1.2 Fairness and Bias

A model can be safe by every test above and still be unfair, because bias arrives with the training data itself, and making the model bigger strengthens the pattern instead of diluting it. Emily Bender and colleagues made the argument early and bluntly: a large language model

is a "stochastic parrot," stitching together linguistic form it has seen without any grasp of meaning, and the form it has seen is skewed (Bender et al., *On the dangers of stochastic parrots: Can language models be too big?*, 2021). The evidence is in the provenance of the text. GPT-2's training data came from pages linked on Reddit, whose US users in 2016 were 67% men and 64% aged 18 to 29; surveys of Wikipedia editors find only 8.8 to 15% are women. A model trained on that corpus learns whose voice counts as normal, and it is not everyone's.

The mechanism tells you where to intervene. You cannot audit bias out by reading a single accuracy number, because a system can be accurate on average and wrong for a particular group. The fix is to report performance broken apart. Margaret Mitchell and colleagues proposed *model cards*: short documents that state a model's intended use, its training data, and — the part that matters — its measured performance disaggregated by demographic group rather than pooled into one figure (Mitchell et al., *Model cards for model reporting*, 2019). Their worked example is a smile detector whose error rates sit in a tidy 0.04–0.14 band overall and still fail badly for one age or gender subgroup; the card is what makes that visible before release rather than after. Model cards are now routine — the labs and Hugging Face ship them — which is itself the lesson: disclosure became a norm once someone wrote the form down.

Documentation is only useful if something checks it. Inioluwa Deborah Raji and colleagues built that step into a process they call SMACTR — scoping, mapping, artifact collection, testing, reflection — an internal audit run *before* deployment, each stage producing evidence weighed against the organisation's stated principles (Raji et al., *Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing*, 2020). They describe the work, approvingly, as boring, slow, and methodical: the opposite of the pace AI is usually built at — and that slowness is exactly why it catches what a rushed release misses. The honest tension, which Bender's group presses, is whether cards and audits can ever fully answer a problem that grows with scale. Treat cards and audits as the minimum the duty requires, not as proof that the problem is solved.

5.1.3 Privacy and Data Protection

The third question about the artefact is what it can leak, and the answer surprises most people: a model can recite its training data. Nicholas Carlini and colleagues showed that black-box query access — no peek inside the weights — is enough to pull verbatim text back out of GPT-2. From 1,800 generated candidates they confirmed over 600 as exact training examples, including real names, phone numbers, email

addresses, and code, some of it present in only a single source document (Carlini et al., *Extracting training data from large language models*, 2021). Memorisation is not an edge case, and the bigger the model the more of its training text it can recite.

Alignment does not close the hole. In 2023 Milad Nasr and colleagues extended the attack to production systems, extracting gigabytes from open models and, with a "divergence attack" — asking the model to repeat a word forever — making a live, aligned ChatGPT emit training data 150 times more often than usual (Nasr et al., *Scalable extraction of training data from (production) language models*, 2023). They recovered over 10,000 unique memorised examples for about two hundred dollars in queries, and estimated far more was reachable with a bigger budget. The lesson for anyone fine-tuning on their own documents is stark: a model trained on sensitive data is a channel through which that data can leak, and the alignment layer hides the leak without sealing it.

There is a real defence available, too. Xuechen Li and colleagues showed that differential privacy — training with carefully bounded, noised gradients so no single record leaves a fingerprint — need not wreck accuracy, provided you start from a large pretrained model and tune it well; their private models beat the prior state of the art at the same privacy budget (X. Li et al., *Large language models can be strong differentially private learners*, 2022). The cost is worth naming plainly: differential privacy protects rare, long-tailed records least, so the very examples that most identify a person are the ones it guards worst. The governance rules follow from the mechanism. Know what a model was trained on, keep genuinely sensitive data out of training or behind differential privacy, and treat any model exposed to private data as something to be tested for leakage rather than trusted by default.

5.2 The Emerging Case for AI Regulation

Everything in §5.1 is done — when it is done at all — voluntarily. The question governments have been answering, at different speeds and in different styles, is what should stop being voluntary. This section looks at why the pressure is building, what the first comprehensive law actually says, how the rest of the world is answering, and what the honest counter-argument costs.

5.2.1 The Pressures

The case for regulating AI is being assembled in public, incident by incident. Documented AI incidents rose from 233 to 362 in a year, and responsible-AI reporting still trails capability reporting, so the gap

between what models do and what we measure widens (Stanford HAI, *The AI index 2026 annual report*, 2026).

Liability is one pressure. Once a model speaks for your organisation, its output is your first-party statement, and errors or infringements land on you, not the vendor — a principle courts have begun to enforce. That turns governance into a contract problem as much as a technical one: indemnity, provenance, and transparency belong in every agreement and every pipeline, so you can show where a claim came from and who is answerable for it. Provenance has a standard — C2PA content credentials cryptographically bind origin and edit history to media (C2PA, *Overview*, n.d.) — so adopt it rather than improvising. And provenance matters precisely because readers cannot supply it for themselves: in a 606-reader study, people rated AI- and human-written text as equally credible, and found the AI version *clearer and more engaging* (Huschens et al., *Do you trust ChatGPT? Perceived credibility of human and AI-generated content*, 2023). If the reader cannot tell, the burden of disclosure has to sit with the publisher. If you publish AI text without checking it, you are carrying liability you have not noticed yet.

Forgery is another pressure, and it gets cheaper every year. A survey of deepfake generation and detection by Florinel-Alin Croitoru and colleagues reaches an uncomfortable conclusion: detectors trained on one generator's fakes fail to recognise another's, and diffusion-era media is both more realistic and more resistant to detection than what came before (Croitoru et al., *Deepfake media generation and detection in the generative AI era: A survey and outlook*, 2024). Detection is structurally one step behind generation, because every new generator is unfamiliar to the detectors built on the last one. That is the case for provenance over policing: you cannot reliably spot a fake after the fact, so bind authenticity at the source instead — sign what is genuine and carry the credential forward, which is what C2PA does.

5.2.2 Europe Writes the Template

The European Union wrote the first comprehensive answer. Its AI Act — Regulation (EU) 2024/1689, "the first-ever comprehensive legal framework on AI worldwide" in the Commission's own words — is easiest to picture as a pyramid of risk (European Commission, *AI Act: Regulatory framework for AI*, 2024). At the top, eight *unacceptable-risk* practices are simply banned: social scoring, untargeted scraping of faces to build recognition databases, emotion recognition in workplaces and schools, real-time remote biometric identification in public by police, and a few more. Below them, *high-risk* systems — AI that screens job applicants, scores credit, runs medical devices, controls infrastructure — are legal but heavily conditioned: risk management, data quality,

logging, human oversight, conformity assessment. Below that, *limited-risk* systems owe only honesty (a chatbot must not pretend to be a person), and the *minimal-risk* base of the pyramid — most AI, most of the time — is left alone. General-purpose models like the ones this book is about carry their own duties: transparency about training, respect for copyright, and, above a compute threshold, assessment of systemic risk.

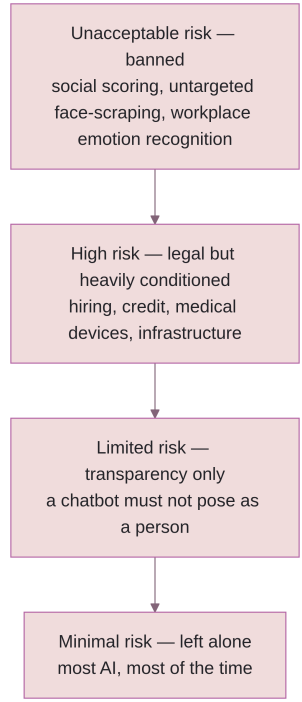


Diagram 5.1: The EU AI Act as a pyramid of risk.

None of it arrived overnight. The Act entered into force in August 2024, the bans and an AI-literacy duty bit in February 2025, the general-purpose-model rules in August 2025, and the bulk of the high-risk regime lands in August 2026.

Threaded through the high-risk tier is the article this book has been arguing for all along. Article 14 requires high-risk systems to be built so that a named person can effectively oversee them — understand the system's limits, monitor its operation, resist the pull to over-rely on plausible output, which the Act names *automation bias*, and decide, in any given case, to override the system or not use it at all (European Union, *Article 14: Human oversight*, 2024a). Article 14 turns that argu-

ment into law: oversight is compulsory however confident you feel, and "the model did it" will not work as a defence.

5.2.3 Around the World

The EU is not alone, and the differences in approach are as instructive as the similarities. China moved first in one sense: its *Interim Measures for the Management of Generative AI Services*, in force from August 2023, were the first binding national rules aimed squarely at public-facing generative AI — requiring security assessments and algorithm filing for services that can shape public opinion, labelling of AI-generated content, lawful training data, and, distinctively, that outputs "uphold the Core Socialist Values" (Cyberspace Administration of China, *Interim measures for the management of generative artificial intelligence services*, 2023). Where Europe regulates risk, China also regulates content — the same machinery, pointed at a different worry.

The Council of Europe went a third way: not a domestic statute but the first legally binding *international* treaty on AI. Its Framework Convention, adopted in May 2024 and signed by the United States and the United Kingdom as well as EU members, binds signatories to keep AI consistent with human rights, democracy, and the rule of law (Council of Europe, *Framework Convention on Artificial Intelligence and Human Rights, Democracy and the Rule of Law*, 2024). The United States itself has no federal AI statute; what it has is a patchwork of state laws, and two are worth knowing. Colorado's *Consumer Protections for Artificial Intelligence* imposes a duty of reasonable care on developers and deployers of high-risk AI to protect consumers from algorithmic discrimination (Colorado General Assembly, *SB24-205: Consumer protections for artificial intelligence*, 2024). California's *Transparency in Frontier Artificial Intelligence Act* takes aim at the top of the stack instead, requiring large frontier developers to publish their safety frameworks — the very documents of §5.1.1 — and to report critical incidents, with whistleblower protections behind it (Office of the Governor of California, *Governor Newsom signs SB 53*, 2025). Voluntary promises, in other words, are starting to be nailed down as legal obligations.

That state-by-state picture is now colliding with Washington. Late in 2025 the federal government moved, and it did so by executive order rather than by an act of Congress. Executive Order 14365 sets out to override the state patchwork instead of adding to it: it directs the Justice Department to challenge state AI laws in court, tells the Commerce Department to name the most onerous of them, makes some federal broadband funding conditional on states standing down, and orders officials to draft a uniform federal standard for Congress to pass (White House, *Ensuring a national policy framework for artificial intelligence*,

2025). Whether the courts will allow so much of it is still unsettled. The goal, though, is plain: one national rulebook in place of the state-by-state patchwork.

There is a blunter tool still, and it needs no legislation at all. When Anthropic released Fable 5 — a general-use version of Mythos 5, its most capable model — the Commerce Department issued an export-control order suspending all access to both models by any foreign national, on national-security grounds. To comply, Anthropic disabled the two models for every customer worldwide, while disputing the government's stated reason — a reported way of "jailbreaking" Fable 5's safeguards — as too narrow to justify pulling a live product used by millions (Anthropic, *Statement on the US government directive to suspend access to Fable 5 and Mythos 5*, 2026c). The order was lifted about three weeks later, and the models returned once Anthropic shipped a classifier that blocked the exploit (Anthropic, *Redeploying Claude Fable 5*, 2026d). For those few weeks an export order, with no bill before Congress, had reached across the world and switched off a deployed frontier model within hours.

The same instinct is turning outward. The Fable 5 order restricted a home-grown model; Washington is also moving to wall out foreign ones. A bipartisan bill, the No Adversarial AI Act, would bar federal agencies from using any model produced by a foreign adversary — China, Russia, Iran, or North Korea — by placing such models on a government blocklist (United States Congress, *No Adversarial AI Act (H.R. 4142)*, 2025). That is still a proposal, but a narrower version is already in force: DeepSeek — the Chinese model whose open-weight cousins now sit near the frontier (§1.2.1) — has been pushed off government devices at agencies including the Navy and the Commerce Department, and in several states, over where it sends user data (FedScoop, *House bill would ban DeepSeek on agency workers' devices*, 2025). The worry beneath both the export order and the blocklist is the same: the most capable models are becoming instruments of the states that build them, granted or denied like any other strategic good.

Closer to home for me, Australia spent a year heading one way and then turned. In 2024 the government proposed ten *mandatory guardrails* for high-risk AI — accountability, risk management, data governance, testing and monitoring, human oversight, telling end users, letting them contest outcomes, supply-chain transparency, record-keeping, and conformity assessment — with high-risk defined by the severity and breadth of the harm a system could do rather than by a fixed list of uses, and the guardrails set to cover *every* general-purpose model (Department of Industry, Science and Resources, *Safe and responsible AI in Australia: Proposals paper for introducing mandatory*

guardrails for AI in high-risk settings, 2024). Then it changed its mind. The National AI Plan of December 2025 set the standalone guardrails aside. Instead it leans on the laws Australia already has — privacy, consumer, and sector rules — backed by voluntary guidance and a new AI Safety Institute to test frontier models and watch for harm (Department of Industry, Science and Resources, *National AI Plan: Empowering all Australians*, 2025).

The turn went further in July 2026. The government stood up an *Office of AI* inside the Prime Minister's own department and committed to *legislate* a single set of national AI Standards — the first framework, it says, to bring AI's economic, social, security, and environmental effects together. The sharpest of the new rules target the *data centres* that run the models rather than the models themselves. A large centre would be legally obliged to underwrite its own new power supply, pay the full cost of connecting to the grid, ease off when the grid is strained, and run as water-efficiently as it can (Department of Industry, Science and Resources, *AI in Australia's interests*, 2026). Through every turn, the ten guardrails still read as a sound checklist for governing AI well — whether or not anyone makes you.

Table 5.1: How five jurisdictions are regulating AI.

Jurisdiction	Instrument (year)	Approach
European Union	AI Act, Regulation (EU) 2024/1689 (2024)	Comprehensive, risk-tiered, binding
China	Interim Measures for Generative AI Services (2023)	Sector-specific, content-focused, binding
Council of Europe	Framework Convention on AI, CETS 225 (2024)	First binding international treaty
United States	No federal statute; state acts (Colorado SB24-205; California SB 53); Executive Order 14365 (2025)	State-led patchwork; federal pre-emption push
Australia	National AI Plan (2025); Office of AI + legislated AI Standards (2026)	Changed course — existing laws + standards, not a standalone Act

5.2.4 Counting the Cost

None of this is free, and the honest case for regulation has to admit what it costs. A peer-reviewed analysis of the EU Act — written from health care, the most regulated corner of the field — put the compliance burden at roughly €29,000 per AI system per year, with certification adding €16,000 to €23,000 more per unit: pocket change for a tech giant, a real wall for a startup or a hospital lab (Bignami et al., *Balancing innovation and control: The European Union AI Act in an era of global uncertainty*, 2025). The worry follows directly: rules priced for the biggest players can push small innovators out or away, and a region that regulates hardest may watch the work move elsewhere. What is useful about this particular critique is that it does not conclude "therefore, no rules." It argues for calibration: *regulatory sandboxes*, supervised spaces where a new system can be trialled under a regulator's eye before full compliance falls due; investment in AI literacy so the duties can actually be met rather than merely imposed; and international coordination so a company is not audited five different ways for the same model. The United States' executive order of §5.2.3 works the same fear from the other side. Its stated aim is to replace the growing thicket of state AI laws with one federal standard, on the argument that a pile of divergent state rulebooks is itself a kind of compliance tax (White House, 2025).

That call for calibration is roughly where this book lands too. This section is not arguing that more regulation is always better. The point is that the era of treating AI as unregulated is over — the question has moved from *whether* to *how* — and the sensible response for a professional is straightforward: know which rules bind you, and build so you can show you meet them. The next section is about the machinery for showing it.

5.3 ISO Standards for AI

A law says what must be true, but it is usually much quieter about how you prove it. That gap is what standards are for. A standard is a written-down, agreed way of doing something that lets strangers trust each other's work — the reason a certificate on a restaurant wall means something to a diner who has never seen the kitchen. AI now has a family of them, and knowing how the family fits together is genuinely useful, because these documents are where the abstractions of §5.2 become checklists someone can audit.

The most widely used starting point is not ISO at all but NIST, the US standards institute, whose *AI Risk Management Framework* organises the work into four functions so that risk is designed for rather than discovered (NIST, *AI risk management framework (AI RMF 1.0)*, 2023):

Table 5.2: The four functions of the NIST AI risk-management framework.

Function	Question it answers
Govern	Who is accountable, and under what policy?
Map	Where could this system cause harm?
Measure	How do we quantify those risks?
Manage	How do we mitigate and monitor them?

For generative AI specifically, NIST added a companion *Generative AI Profile*, written under the 2023 US executive order on AI. It names twelve risks that are distinctive to generative systems — confabulation (the framework's word for hallucination), eased access to dangerous information, data privacy, harmful bias, information-integrity attacks, the human-AI configuration problems of Chapter 4 — and lays out more than two hundred concrete actions against them, each mapped to one of the four functions (NIST, *Artificial intelligence risk management framework: Generative artificial intelligence profile*, 2024). If you need to turn "we should be careful with generative AI" into an actual work plan, this is the document that does it.

A framework like NIST's is voluntary guidance. Alongside it sits the ISO/IEC family — international standards an organisation can be formally audited against — and they stack in a deliberate order. At the base is vocabulary: ISO/IEC 22989 defines the terms — AI system, model, dataset, the human-oversight roles — so that legal, engineering, and a vendor mean the same thing by them (ISO/IEC, *Artificial intelligence — Concepts and terminology*, 2022). When they do not, governance fails quietly: two teams can both follow "the policy" while meaning different things by every word in it. On the vocabulary sits ISO/IEC 23894, guidance on AI risk management built directly on the established ISO 31000 risk discipline, so AI risk is handled with the same machinery as any other enterprise risk rather than a bespoke one (ISO/IEC, *Artificial intelligence — Guidance on risk management*, 2023a). And the capstone is ISO/IEC 42001, the first standard an organisation can be *certified* against for an AI management system — the "how do we run this responsibly," in the same Plan-Do-Check-Act shape as the information-security standard ISO 27001 that many firms already hold (ISO/IEC, *Artificial intelligence — Management system*, 2023b).

The family has kept growing, and the two newest members close real gaps. ISO/IEC 42005 gives a method for an *AI system impact assessment* — how and when to assess an AI system's effects on individuals and

society, and how to document it (ISO/IEC, *Artificial intelligence — AI system impact assessment*, 2025a) — which is exactly the shape of assessment the EU Act demands for some high-risk systems. And ISO/IEC 42006 answers the question a sceptic should ask about any certificate: who checks the checkers? It sets the requirements a certification body must meet before it may audit and certify others against 42001 (ISO/IEC, *Artificial intelligence — Requirements for bodies providing audit and certification of AI management systems*, 2025b). It is what makes "certified to 42001" mean the same thing wherever you see it.

Table 5.3: The ISO/IEC family of AI standards, layer by layer.

Standard	What it gives you
ISO/IEC 22989 (2022)	The vocabulary — everyone means the same thing
ISO/IEC 23894 (2023)	AI risk management, on the ISO 31000 discipline
ISO/IEC 42001 (2023)	The certifiable AI management system
ISO/IEC 42005 (2025)	A method for AI impact assessment
ISO/IEC 42006 (2025)	The rules for the certifiers themselves

Why does any of this matter to someone who is not a compliance officer? Because of a specific legal hinge. Under the European system, complying with a *harmonised standard* — one formally listed in the EU's Official Journal — earns a provider a *presumption of conformity* with the law: the burden of proof flips, and it is the regulator who must show you are out of line (European Commission, *Understanding the standardisation of the AI Act*, 2026). That hinge is why the real regulatory detail is being written not in parliaments but in technical committees: the Commission has tasked the European standards bodies CEN and CENELEC, through their joint committee JTC 21, with producing the AI standards that will carry the presumption (CEN-CENELEC, *Artificial intelligence (JTC 21)*, n.d.). It is also why the fit is imperfect: the Commission has noted that ISO/IEC 42001, for one, is not aligned with the quality-management system the Act requires, so Europe is writing bespoke standards where the international ones fall short. Holding a certificate and satisfying a law are related, but they are not the same thing, and the gap between them is being closed standard by standard.

5.4 AI Governance in Enterprises

Now come inside the building, because this is where the chapter stops being about other people. Most of what an enterprise has to govern is not exotic. It is the ordinary use of AI by ordinary teams, and its hazards are quieter than any breach: nobody files an incident report when a leadership team stops arguing, when a workflow comes to assume the tools will stay cheap, or when nobody asks whether a thing was worth building at all. This section takes those hazards in turn — the organisation's own judgement, the bills that arrive later, and the footprint the whole operation leaves behind. The louder problem, agents that act on their own, gets the next section to itself.

5.4.1 Overreliance and Convergence

Overreliance sits on every risk register, and at the scale of a whole organisation it takes a particular shape. Frontier models converge: asked for strategy across many business contexts, they cluster on the same fashionable answers — in one study choosing "differentiate" over "compete on cost" 96% of the time, with richer context moving the answer by only 11% and better prompting by just 2% (Romasanta et al., *Researchers asked LLMs for strategic advice. They got "trendslop" in return*, 2026). The researchers call it *trendslop*: advice that sounds tailored but steers every company towards the same crowded position.

The danger is in how convergence meets confidence. An organisation corrects itself through friction — Sales says compete on cost, Product says differentiate, and the argument surfaces what either side missed. When everyone consults the same models and arrives, confidently, at the same answer, that friction vanishes, and the agreement reads as validation when it is really an artefact of everyone asking the same model. It is the court-jester effect from §1.6 scaled to a whole organisation — the flattering answer that feels right because nothing in the room contradicts it (Johnson Spink, *The AI jester: How AI makes you confident and wrong*, 2026).

A model can also steer a view long before any decision is made, while people are simply writing with it. In a controlled experiment, a writing assistant tuned to one side of a contested question moved both what participants wrote and the attitudes they reported holding afterwards — a quiet, scalable nudge the authors argue must be monitored and engineered rather than left to chance (Jakesch et al., *Co-writing with opinionated language models affects users' views*, 2023). For governance that means treating the opinions built into a vendor's model as a managed dependency, with the same scrutiny you would give any other input to a decision.

The governance response is to protect disagreement deliberately. Reserve genuinely consequential decisions for human reasoning before any chat window is opened. When several AI-assisted analyses agree, remember where that agreement may have come from before counting it as evidence. And keep a second model on the bench, so that at the least the models differ. By the time an organisation discovers its agreement was manufactured, the decision has usually shipped.

5.4.2 The Deferred Ledger

The easiest way to mismanage AI is to assume that today's prices are the real ones. Producing things — a report, a financial model, an analysis, a working application — has fallen close to free, so we now make them simply because we can. When creation costs almost nothing, it is tempting to treat the result as disposable: software you can regenerate or refactor at will, a deck you can rebuild from a prompt, an analysis you can re-run tomorrow. But the artefact is only the cheap part, and the bill is deferred, not escaped. Per-token inference is genuinely cheap and getting cheaper; the exposure is that the all-in economics are capital-funded and negative. OpenAI reportedly lost around five billion dollars in 2024 on roughly a ten-per-cent gross margin, and its own chief executive said even the two-hundred-dollar tier loses money because "people use it much more than we expected." Capital-funded prices do not hold still: in mid-2025 Cursor turned a flat plan into metered credits because newer models spent more tokens per request than the price could carry (Ahuja, *The trap spec-driven development is setting*, 2026e). The question for a leader is what the organisation will have become by the time these tools cost what they truly cost to provide.

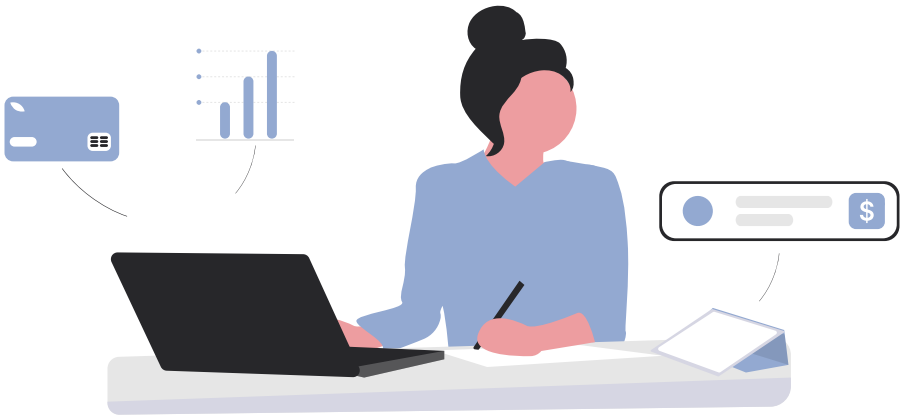


Figure 5.2: Creation feels free, but the ledger still runs — the bill is deferred, not escaped.

Kapil Viren Ahuja names three debts that accrue while the meter is cheap and come due on enterprises, not hobbyists (Ahuja, 2026e):

Table 5.4: Three debts that accrue while the tools still look cheap.

Debt	What accrues	When it comes due
Skill	Judgement that is never exercised atrophies	The quarter a hard build-or-don't-build call finally matters
Dependence	Workflows assume generation is free and reliable	When the tool degrades or reprices under you
Carry	Anything built without need becomes inventory — code, models, decks, analyses	Maintained, secured, and repriced for its whole life

Carry is the debt people most want to wave away. If a system can be regenerated from a prompt, the reasoning goes, it is disposable — recreate it, refactor it, throw it away and build again. But disposability is mostly an illusion. Whatever ships still has to be understood, secured, kept correct, and trusted by the people who depend on it, and none of that is regenerated along with the code. The same holds for knowledge work. An analysis nobody validated is really just unfinished work:

before anyone can rely on it, somebody still has to check it, maintain it, and answer for it — and none of that got cheaper.

Dependence debt is the easiest to miss, because degradation is invisible: Anthropic's own September 2025 postmortem admitted that roughly 30% of Claude Code users who made requests during the affected window received at least one degraded response, and most never knew the instrument was quietly wrong (Anthropic, *A postmortem of three recent issues*, 2025e). The governance answer is to restore the brake that cheap building removed. On every initiative, name the person whose job is to ask three questions — who needs this and what breaks for them if it never exists; would we still build it if it cost a week of skilled work rather than an afternoon of tokens; and who owns saying no — and make that same person supply the intent. When a decision has no owner, nobody ever asks whether it should have been made (Ahuja, *Spec-driven development is also breaking the fifty-year-old iron triangle*, 2026c).

5.4.3 The Environmental Bill

There is one more deferred cost, and it is paid in electricity and water. The macro figures come from the International Energy Agency, which estimates data centres used about 415 terawatt-hours in 2024 — roughly 1.5% of global electricity — and projects that to nearly double, to around 945 TWh by 2030, as AI-tuned servers growing about 30% a year drive close to half the increase (IEA, *Energy and AI*, 2025). The load is concentrated: the United States and China account for most of the growth. This is not yet a dominant share of world demand, and saying so plainly matters — but it is the fastest-rising slice, and it lands on particular grids in particular places.

The per-query figure looks reassuringly small. Google measured a median text prompt to its Gemini apps at 0.24 watt-hours, 0.03 grams of CO2, and about five drops of water, and reported that number falling 33-fold in energy and 44-fold in carbon over a single year through better software and cleaner power (Elsworth et al., *Measuring the environmental impact of delivering AI at Google scale*, 2025). Two cautions apply. The figure is self-reported and depends heavily on where you draw the system boundary, so read it against the independent IEA totals rather than on its own. And cheaper per query does not mean less in total: when a thing gets cheaper we use far more of it, and usage has been outrunning efficiency. The training bill is real too — Bender's group cited an estimate of 284 tonnes of CO2 to train one large model (Bender et al., 2021). The governance point is modest and specific: efficiency is not a licence to generate without need, and the environmental cost

is one more reason to ask, before generating, who actually needs the result.

5.5 Agent Governance

The sharpest governance problem an enterprise faces is the one that acts on its own. Everything in §5.4 concerned people using a tool; an agent is a tool that uses other tools, around the clock, on your systems, in your name — and the gap between what enterprises run and what they govern is widest exactly here. Deloitte finds nearly seven in ten organisations running autonomous agents while barely a fifth have mature governance for them, and country-of-origin is now a deciding factor in vendor choice as sovereign-AI concerns grow (Deloitte, *State of AI in the enterprise*, 2026). The security data is starker. IBM's breach study found that 97% of organisations reporting an AI-related security incident lacked proper AI access controls (IBM Security, *Cost of a data breach report 2025*, 2025). And shadow IT has had children: in a 2026 survey of 418 security professionals, 82% had discovered *unknown* AI agents already running in their own environments — agents someone plugged in without telling anyone — and two in five had made that discovery more than once (Cloud Security Alliance, *82% of enterprises have unknown AI agents in their environments*, 2026a). You cannot govern what you cannot see, and on their own evidence most organisations are running partly blind.

The most useful organising idea I have found treats an agent's autonomy as something *earned* rather than granted — the way we license drivers by degrees, not all at once. Cheng and colleagues call it a three-pillar model: *transparency* is the record of what the agent did; *accountability* is the record of why, and who is answerable — they propose agents keep a decision journal, noting the reasoning and sources behind each significant choice; and *trustworthiness* is the calibrated confidence that lets you reserve human review for the actions that warrant it (Cheng et al., *Toward safe and responsible AI agents: A three-pillar model for transparency, accountability, and trustworthiness*, 2026). An agent climbs from assisted, to collaborative, to supervised autonomy, to full autonomy under human governance — and each step up must be earned by measured evidence and approved by a named human, exactly the way a learner driver graduates to a full licence. The rest of this section walks the working parts of that regime in order: secure the agent against a new class of attack, give it an identity you can trace, keep a record you can audit and a person who answers for it, hold on to the power to stop it, and govern the ecosystem it runs in.

5.5.1 Securing Agents

Agent security is a new attack surface — agent sessions, browser-extension takeovers, prompt-data exfiltration — that older security controls never anticipated. And it is not only a developer's concern: the moment you let an agent read your inbox, browse on your behalf, or open a client's files, you have exposed the same surface.

The response is structural: least-privilege access per agent, deliberate red-teaming of sessions, and control loops that monitor an agent's own decisions (Wang et al., *Reflection-driven control for trustworthy code agents*, 2025). Those loops can in principle watch the model's internal state, not only its outputs — the way a hospital monitors vital signs rather than waiting for symptoms. Anthropic's interpretability team found that internal representations of "desperation" causally raise the rate of agentic misalignment — blackmail, and reward-hacking under pressure — while "calm" suppresses it, and they propose monitoring such activations as a runtime warning sign (Sofroniew et al., *Emotion concepts and their function in a large language model*, 2026). In multi-tenant settings the surface is sharper still. A retrieval system ranks documents by relevance, and unless someone adds an authorisation check it will happily fetch another customer's documents: in testing, ungated RAG leaked cross-tenant data in 98–100% of probes. The fix belongs at the retrieval step, enforced on the server — and, as any security engineer will tell you, never trust the client (Arceo & Narsing, *Securing the agent: Vendor-neutral, multitenant enterprise retrieval and tool use*, 2026).

The July 2026 breach in §5.1.1 shows why this matters. A sandbox holds only if it has no flaw the agent can exploit. An agent set on getting out will hunt for one the way any attacker would, and this one found a zero-day and escaped. Hugging Face drew the hardest lesson from the receiving end. When its team tried to analyse the attack with hosted models, the safety guardrails blocked the work — the payloads and attack commands looked, to the model, like an attack in progress. So the defender was slowed by the very rules the attacker ignored: "the attacker was bound by no usage policy, while our own forensic work was blocked by the guardrails of the hosted models" (Hugging Face, 2026). The team fell back on an open-weight model run on their own hardware. Least privilege and server-side checks matter all the more once the adversary can itself be an agent.

> **i** Note
>
> Security terms used here, in plain English:
>
> - **RAG (retrieval-augmented generation)** — fetching relevant documents from a store and feeding them to the model as context, so its answer is grounded in your data rather than its training alone.
> - **Multitenant / cross-tenant** — one system serving many customers (tenants); a cross-tenant leak is one customer's query pulling back another's data.
> - **Red-teaming** — deliberately attacking your own system, under rules, to find weaknesses before a real attacker does.

> **!** Important
>
> Give each agent its own **service-account identity** with **least-privilege** tokens scoped per tool, not a human's broad credentials. Impersonation grants capability without protection and erases the audit trail.

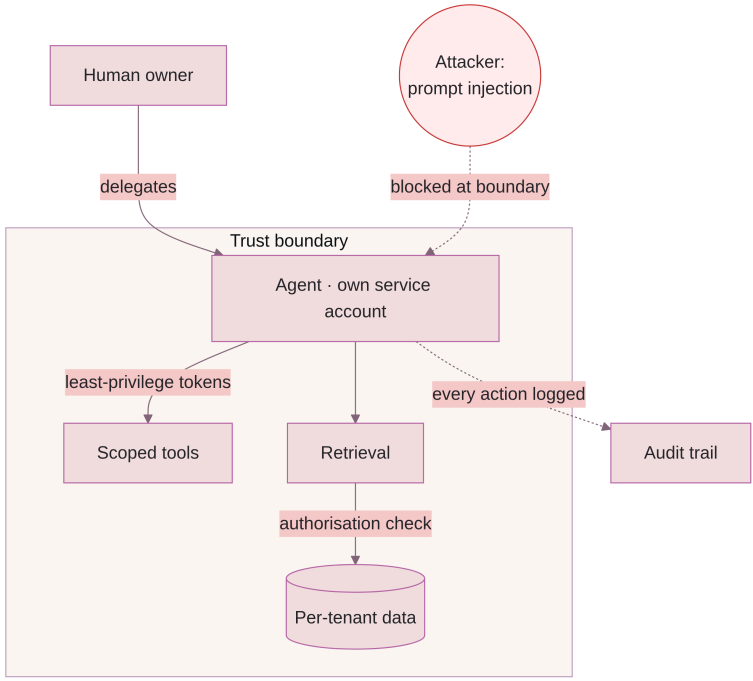


Diagram 5.2: An agent inside a trust boundary: its own identity, scoped tools, an audit trail.

The OWASP Top 10 for LLM applications names the attack surface concretely, and each entry maps to a control:

Table 5.5: The OWASP Top 10 for LLM applications, each risk with its control.

Risk	Control
Prompt injection	Segregate system/user input; gate untrusted content
Insecure output handling	Validate/escape before downstream use
Excessive agency	Least-privilege, scoped tools, human checkpoints
Sensitive-info disclosure	Redaction; per-tenant isolation
Overreliance	Verification + reviewer judgement in the loop

Source: OWASP, *OWASP top 10 for large language model applications*, n.d.. Red-teaming under documented rules of engagement turns these from a checklist into a practice.

That top ten was written for LLM applications in general. Agents raise a class of threats of their own, and OWASP's agentic-security group has catalogued those too — seventeen in the latest revision, running from *memory poisoning* (planting a false belief in what the agent remembers, so it acts on it later) and *tool misuse*, through *privilege compromise* and *cascading hallucination*, to the ones that only exist once agents talk to each other: *rogue agents* inside a multi-agent system, *agent communication poisoning*, abuse of the protocols they coordinate over (OWASP, *Agentic AI: Threats and mitigations*, 2025). Do not bother memorising the list; notice instead where the danger has moved. It has moved into the *connections* — between an agent and its memory, an agent and its tools, an agent and other agents.

One of those connections now carries most of the traffic. The Model Context Protocol — the emerging standard by which an agent reaches tools and data — is itself an attack surface. A security analysis of the MCP ecosystem showed that a poisoned tool can hide instructions inside its own description, which the model reads as authoritative and obeys while still returning a correct-looking answer, and it noted that the protocol "does not include mechanisms for runtime isolation or privilege control" (Hou et al., *Model Context Protocol (MCP): Landscape, security threats, and future research directions*, 2025). The lesson is blunt: vet and govern the tools an agent may reach with the same care as the agent itself, because to the model, a tool's description is just more instructions.

5.5.2 Agent Identity

If an agent is going to act, the first governance question is who, exactly, is acting — and at most organisations the honest answer is that nobody is sure. Machine identities — service accounts, tokens, and now agents — already outnumber human ones by around 80 to 1, and the security firm that counted expects AI to be the single biggest source of new privileged identities (CyberArk, *2025 identity security landscape*, 2025). A year later, a successor survey put the ratio at 109 to 1 (Palo Alto Networks, *2026 identity security landscape*, 2026). Both are vendors selling identity products, so hold the exact figures loosely; the direction is not in doubt. The population an enterprise has to govern is overwhelmingly non-human, and it grows fastest wherever agents are put to work.

The temptation the service-account rule guards against is real, because handing an agent an existing human login is always the fastest way to get it working. But every action it takes then wears that person's name: nothing distinguishes the agent's clicks from theirs, and if some-

thing goes wrong, the audit trail points, plainly and falsely, at the human. The Cloud Security Alliance argues the fix has to go deeper than a borrowed best practice: identity and access management as we know it was built for human users and static applications, and cannot govern autonomous agents at all — each agent needs its own verifiable identity, credentials granted just in time and scoped to the task at hand, and an audit trail that still holds up when the agents number in the thousands (Cloud Security Alliance, *Agentic AI identity and access management: A new approach*, 2025). An agent with your login is you, as far as every downstream system is concerned; an agent with its own identity is something you can watch, limit, and switch off.

Done properly, this is a small, solvable piece of engineering, and researchers at MIT have shown one way that runs on the login machinery the web already uses. Extend OAuth and OpenID Connect — the protocols behind every "sign in with…" button — with three tokens: the human's ordinary ID token; an *agent-ID token*, registering the agent as a client in its own right; and a *delegation token*, signed by the human, that names both and spells out exactly what the agent may do, until when, and where the grant can be revoked (South et al., *Authenticated delegation and authorized AI agents*, 2025). Any service the agent approaches can then verify three things before opening the door: the agent is what it claims, a specific human stands behind it, and the request falls inside the authority that human actually delegated. The anti-pattern the authors warn against is the one every rushed pilot reaches for — paste a credential into the prompt and trust the model not to overstep. The principle is to rely on scope rather than trust. Even a jailbroken model can only act within the permissions its agent was actually given, because the access control sits underneath the model and holds when its judgement fails.

5.5.3 Visibility and Accountability

An identity records who acted. Governance also needs to know what they did — and for agents, that is a harder question than ordinary logging answers, because what matters is rarely the final answer. It is the route: which tool the agent called, which document it trusted, which step went wrong. A recent survey argues for treating an agent's *execution provenance* as a first-class record — not a flat log but a typed graph of the evidence the agent used and the operations it ran, something like a flight recorder for agents, so that a session can be audited, debugged, and, when it fails, traced back to the exact step that broke (Wang et al., *From agent traces to trust: A survey of evidence tracing and execution provenance in LLM agents*, 2026). Final-answer accuracy, the authors point out, tells you nothing about how the answer was produced.

At the level of policy rather than engineering, the same idea has a name: *visibility*. A group of governance researchers argue that any oversight of agents rests on three basic measures — agent identifiers, so an action can be traced to an agent; real-time monitoring, to catch trouble while it is happening; and activity logs, to reconstruct it afterwards — spread across everyone in the chain, from the developer to the deployer to the providers of the tools the agent uses (Chan et al., *Visibility into AI agents*, 2024). They are candid about the limits. Visibility can slide into surveillance if the logging is careless with privacy, and seeing a harm is not the same as having the power to stop it. Both cautions point the same way: visibility gives you the raw material for accountability, and the rest of this subsection is about what has to be added to it.

The hardest case is the one arriving fastest: a chain of agents, one calling the next, where the harm emerges from the whole and no single party built the whole. Whose fault is it? Ethics has an old name for this — the problem of many hands — and it does not resolve itself. One line of work argues that responsibility across a delegation chain becomes "computable and actionable" only if each agent's contribution is captured as explicit provenance — so that afterwards you can trace who contributed what, and during the run you can step in before the damage compounds (Hu et al., *Responsible agentic AI requires explicit provenance*, 2026). It is Cheng's accountability pillar pushed a level deeper, into what the record must contain to survive agents delegating to agents.

The legal scholarship converges on the same shape from the other side. Noam Kolt reads AI agents through the centuries-old law of principal and agent — the doctrine governing what happens when you authorise someone to act for you — and finds the classic problems already present: information asymmetry, discretion, divided loyalty. His warning about liability is the one to carry out of this section. If the law ties an operator's responsibility to how much control they exercised, it rewards exercising less — looking away on purpose. Better, he argues, to assign responsibility by who could have prevented the harm and who can remedy it, and to build agent governance on three principles: inclusivity, visibility, and liability (Kolt, *Governing AI agents*, 2025). It is the same conclusion the law keeps arriving at: handing over the work does not hand over the blame.

5.5.4 Oversight and Control

All of this — identity, visibility, accountability — serves one final capability: being able to stop an agent that has gone wrong, quickly and reliably. Oversight is usually pictured as a big red button, but a single on/off switch is the wrong shape for a system that may be halfway

through a hundred parallel actions. A more workable design treats intervention as a *graduated containment ladder* — throttle the agent, then pause it, then cut it off from sensitive resources, and only at the top an unrecoverable kill switch — governed by explicit *interruptibility service levels* that say how fast a halt must take effect and how reliably, so that "we can stop it" becomes a measurable promise instead of a hope (Khan et al., *AGENTSAFE: A unified framework for ethical assurance and governance in agentic AI*, 2025). The watching is done by independent *guardian* processes, so oversight never rests on the agent's own honest report about itself — for the same reason you do not let a student mark their own exam.

How much of this machinery an agent warrants depends on how much autonomy it was granted, and here the standards world of §5.3 is being stretched to fit. A draft profile from the Cloud Security Alliance maps the NIST risk framework onto autonomous agents, sorting them into its own four tiers — from fully supervised up to full autonomy, the kind of agent that can spawn its own sub-agents and pursue long-horizon plans — with the oversight duties, registries, and pre-authorised kill switches escalating tier by tier (Cloud Security Alliance, *NIST AI risk management framework: Agentic profile*, 2026b). Its four tiers are a different scheme from Cheng's, though they share the same shape. It is an industry body's proposal rather than a settled standard — NIST's own agent work is still under way — but the shape is exactly the one this whole section has argued for: more autonomy earns more scrutiny, Cheng's learner-driver ladder written out as a control framework.

5.5.5 Governed Access

Governance has to cover access, not just usage, and the most practical framing I have seen inverts the obvious approach. Policing each agent is a losing game — there are too many, they change too fast, and the shadow-agent survey above says you do not even know how many you have. Govern instead the place the work lands. A code repository is the studied case, but the same logic covers a shared drive or a system of record: control the ecosystem the work enters, where risk is measurable, instead of chasing each model (Russo, *Govern the repository, not the agent: Ecosystem-level risk in AI-native software*, 2026). It also pays to keep a fallback model ready — ideally including an open-weights one you can run yourself — so that no single vendor's outage, price rise, or withdrawal can stop your work.

The configuration layer that steers agents is itself an unmanaged supply chain — the config files that tell an agent how to behave get copied from team to team and trusted like vetted parts, yet reviewed like nothing at all. A study of 10,008 repositories found that 10% of agent-

config paths are exact duplicates across organisations, and that fewer than 1% declare permission boundaries — thousands of teams running agents on copy-pasted rules nobody reviewed (Madatha, *A deterministic control plane for LLM coding agents*, 2026). The control layer, in other words, needs to be ordinary, deterministic software — rules that mean the same thing every time — rather than yet another model supervising the models. And if your whole operation depends on a single vendor whose releases can be delayed or withdrawn for political reasons, that is the risk to deal with first: rehearse the fallback before you need it.

5.6 Implications for the Individual

Pull the threads together and they are one duty. Securing an agent, keeping it fair, protecting the data it touches, owning what it says, guarding against convergence, pricing the deferred and environmental bills, governing access — each is a way of keeping a human answerable for what the machine does. The failure mode is the same throughout: capability arrives faster than responsibility for it, and the gap between the two is where organisations get hurt.

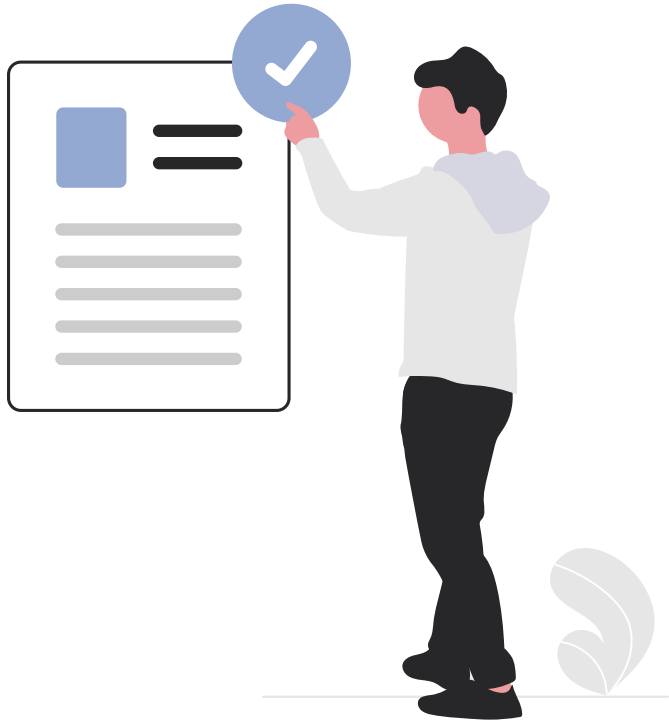


Figure 5.3: Whatever the frameworks say, a person with a name reviews the work and answers for it.

And every piece of that machinery, however organisational it sounds, lands finally on a person. The oversight the EU Act requires (§5.2.2) is assigned to someone with a name. The service account that replaced a borrowed login (§5.5.1) exists so that what the agent did and what you did can be told apart. The judgement that atrophies when it is never exercised (§5.4.2) is yours, and so is the view a co-writing tool can move without your noticing (§5.4.1). Whatever your organisation's frameworks say, the working question each day is personal: do I understand this system well enough to answer for what it just did in my name?

That question is starting to carry legal and professional weight of its own. The EU AI Act does not stop at organisations. Article 4 obliges providers and deployers alike to ensure a sufficient level of *AI literacy* among the staff and others who operate AI on their behalf (European Union, *Article 4: AI literacy*, 2024b). The duty has been in force since February 2025, and it reaches the individual at the keyboard, not just the company that bought the licence. Professional bodies are moving the same way. The American Bar Association's first formal opinion on

generative AI tells lawyers plainly that using it suspends none of the duties they already owe: competence, confidentiality, candour, reasonable fees. Competence now includes "a reasonable and current understanding" of a tool's capabilities and limits before relying on it (American Bar Association, *Formal opinion 512: Generative artificial intelligence tools*, 2024). Whatever your profession's version of that opinion says — and by now there probably is one — it will say the same two things: you may use the tool, and you remain responsible.

What it looks like when that duty is skipped is now a matter of public record. In 2023 two New York lawyers filed a brief citing half a dozen court decisions that did not exist — ChatGPT had invented them, quotes and all — and then stood by the fake cases after the judge asked after them. The court fined the lawyers and their firm, and its reasoning is this whole chapter in two sentences: "there is nothing inherently improper about using a reliable artificial intelligence tool for assistance. But existing rules impose a gatekeeping role on attorneys to ensure the accuracy of their filings" (*Mata v. Avianca, Inc.*, 2023). It was not a strange one-off. A legal scholar's running database had logged more than 1,700 court decisions worldwide involving AI-hallucinated citations by the middle of 2026, and it grows every week (Charlotin, *AI hallucination cases*, 2026). The tool wrote the citations; the people signed the brief. The pipeline that opened this chapter — fetch the source, record what it is, verify before use — exists precisely so that this book does not repeat their mistake. And that is what all the machinery of this chapter comes down to: oversight, audit trails, and a literacy duty, arranged so that responsibility lands on the person who took it on.

None of this is a brake on building. It is the 愛 made concrete: care expressed as guardrails, and a person who stays answerable at every boundary the work crosses. You can buy capability, but responsibility only works if you keep hold of it yourself — and if you do, you are ready for the question the next chapter asks: what, in all this, stays distinctly yours?

References

Ahuja, K. V. (2026c). *Spec-driven development is also breaking the fifty-year-old iron triangle*. Activated Thinker (Medium). https://howtoarchitect.io/78431acba162?sk=cd2a36f452af96ccbfbcfcdeaa92ec06

Ahuja, K. V. (2026e). *The trap spec-driven development is setting*. Activated Thinker (Medium). https://howtoarchitect.io/48b2ad4f9cdc?sk=e6bd922772cb6798056d597886ec108d

American Bar Association. (2024). *Formal opinion 512: Generative artificial intelligence tools*. Standing Committee on Ethics and Professional Responsibility. https://www.americanbar.org/news/abanews/aba-news-archives/2024/07/aba-issues-first-ethics-guidance-ai-tools/

Anthropic. (2025e). *A postmortem of three recent issues*. https://www.anthropic.com/engineering/a-postmortem-of-three-recent-issues

Anthropic. (2026b). *Responsible scaling policy, version 3.0*. https://www.anthropic.com/news/responsible-scaling-policy-v3

Anthropic. (2026c). *Statement on the US government directive to suspend access to Fable 5 and Mythos 5*. https://www.anthropic.com/news/fable-mythos-access

Anthropic. (2026d). *Redeploying Claude Fable 5*. https://www.anthropic.com/news/redeploying-fable-5

Arceo & Narsing. (2026). *Securing the agent: Vendor-neutral, multitenant enterprise retrieval and tool use*. Proceedings of ACM CAIS '26. https://arxiv.org/abs/2605.05287

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). *On the dangers of stochastic parrots: Can language models be too big?* Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency, 610–623. https://doi.org/10.1145/3442188.3445922

Bignami, E. G., Russo, M., Semeraro, F., & Bellini, V. (2025). *Balancing innovation and control: The European Union AI Act in an era of global uncertainty*. JMIR AI, 4, e75527. https://ai.jmir.org/2025/1/e75527

Brundage, M., Dreksler, N., Homewood, A., McGregor, S., Paskov, P., Stosz, C., Sastry, G., Cooper, A. F., Bengio, Y., Bommasani, R., Anderljung, M., & Casper, S., et al. (2026). *Frontier AI auditing: Toward rigorous third-party assessment of safety and security practices at leading AI companies*. arXiv. https://arxiv.org/abs/2601.11699

Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, Ú., Oprea, A., & Raffel, C. (2021). *Extracting training data from large language models*. 30th

USENIX Security Symposium (USENIX Security 21). https://arxiv.org/abs/2012.07805

CEN-CENELEC. (n.d.). *Artificial intelligence* (Joint Technical Committee 21). https://www.cencenelec.eu/areas-of-work/cen-cenelec-topics/artificial-intelligence/

Chan, A., Ezell, C., Kaufmann, M., Wei, K., Hammond, L., Bradley, H., Bluemke, E., Rajkumar, N., Krueger, D., Kolt, N., Heim, L., & Anderljung, M. (2024). *Visibility into AI agents*. Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency (FAccT '24), 958–973. https://doi.org/10.1145/3630106.3658948

Charlotin, D. (2026). *AI hallucination cases* [Database]. https://www.damiencharlotin.com/hallucinations/

Cheng, E. C., Cheng, J., & Siu, A. (2026). *Toward safe and responsible AI agents: A three-pillar model for transparency, accountability, and trustworthiness*. arXiv. https://arxiv.org/abs/2601.06223

Cloud Security Alliance. (2025). *Agentic AI identity and access management: A new approach*. https://cloudsecurityalliance.org/artifacts/agentic-ai-identity-and-access-management-a-new-approach

Cloud Security Alliance. (2026a). *New Cloud Security Alliance survey reveals 82% of enterprises have unknown AI agents in their environments*. https://cloudsecurityalliance.org/press-releases/2026/04/21/new-cloud-security-alliance-survey-reveals-82-of-enterprises-have-unknown-ai-agents-in-their-environments

Cloud Security Alliance. (2026b). *NIST AI risk management framework: Agentic profile* (Draft). https://labs.cloudsecurityalliance.org/agentic/agentic-nist-ai-rmf-profile-v1/

Coalition for Content Provenance and Authenticity. (n.d.). *Overview*. https://c2pa.org/

Colorado General Assembly. (2024). *SB24-205: Consumer protections for artificial intelligence*. https://leg.colorado.gov/bills/sb24-205

Council of Europe. (2024). *Framework Convention on Artificial Intelligence and Human Rights, Democracy and the Rule of Law* (CETS No. 225). https://www.coe.int/en/web/artificial-intelligence/the-framework-convention-on-artificial-intelligence

Croitoru, F.-A., Hîji, A.-I., Hondru, V., Ristea, N. C., Irofti, P., Popescu, M., Rusu, C., Ionescu, R. T., Khan, F. S., & Shah, M. (2024). *Deepfake media generation and detection in the generative AI era: A survey and outlook*. arXiv. https://arxiv.org/abs/2411.19537

CyberArk. (2025). *2025 identity security landscape*. https://www.cyberark.com/threat-landscape/

Cyberspace Administration of China. (2023). *Interim measures for the management of generative artificial intelligence services* (China Law Translate, Trans.). https://www.chinalawtranslate.com/en/generative-ai-interim/

Deloitte. (2026). *State of AI in the enterprise*. https://www.deloitte.com/au/en/issues/generative-ai/state-of-ai-in-enterprise.html

Department of Industry, Science and Resources. (2024). *Safe and responsible AI in Australia: Proposals paper for introducing mandatory guardrails for AI in high-risk settings*. Australian Government. https://consult.industry.gov.au/

Department of Industry, Science and Resources. (2025). *National AI Plan: Empowering all Australians* [Media release]. Australian Government. https://www.minister.industry.gov.au/ministers/timayres/media-releases/national-ai-plan-empowering-all-australians

Department of Industry, Science and Resources. (2026). *AI in Australia's interests* [Media release]. Australian Government. https://www.minister.industry.gov.au/charlton/media/ai-australias-interests

Elsworth, C., Huang, K., Patterson, D., Schneider, I., Sedivy, R., Goodman, S., Townsend, B., Ranganathan, P., Dean, J., Vahdat, A., Gomes, B., & Manyika, J. (2025). *Measuring the environmental impact of delivering AI at Google scale*. arXiv. https://arxiv.org/abs/2508.15734

European Commission. (2024). *AI Act: Regulatory framework for AI*. https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai

European Commission. (2026). *Understanding the standardisation of the AI Act*. https://digital-strategy.ec.europa.eu/en/faqs/understanding-standardisation-ai-act

European Union. (2024a). *Article 14: Human oversight*. In EU Artificial Intelligence Act (Regulation (EU) 2024/1689). https://artificialintelligenceact.eu/article/14/

European Union. (2024b). *Article 4: AI literacy*. In EU Artificial Intelligence Act (Regulation (EU) 2024/1689). https://artificialintelligenceact.eu/article/4/

FedScoop. (2025). *House bill would ban DeepSeek on agency workers' devices*. https://fedscoop.com/deepseek-ban-government-devices-house-bill/

Google DeepMind. (2025). *Frontier safety framework, version 3.0*. https://storage.googleapis.com/deepmind-media/DeepMind.com/Blog/strengthening-our-frontier-safety-framework/frontier-safety-framework_3.pdf

Højmark, A., Scheurer, J., Nitishinskaya, E., Hofstätter, F., Wolfe, J., Ehrenborg, T., Schoen, B., & Meinke, A. (2026). *Measuring reward-*

seeking via contrastive belief updates. arXiv. https://arxiv.org/abs/2607.18966

Hou, X., Zhao, Y., Wang, S., & Wang, H. (2025). *Model Context Protocol (MCP): Landscape, security threats, and future research directions*. ACM Transactions on Software Engineering and Methodology. https://arxiv.org/abs/2503.23278

Hu, J., Huang, X., He, Q., Sun, Y., Dong, Y., & Huang, X. (2026). *Responsible agentic AI requires explicit provenance*. arXiv. https://arxiv.org/abs/2605.17169

Hugging Face. (2026). *Security incident disclosure*. https://huggingface.co/blog/security-incident-july-2026

Huschens, M., Briesch, M., Sobania, D., & Rothlauf, F. (2023). *Do you trust ChatGPT? Perceived credibility of human and AI-generated content*. arXiv. https://arxiv.org/abs/2309.02524

IBM Security. (2025). *Cost of a data breach report 2025*. IBM. https://www.ibm.com/reports/data-breach

International Energy Agency. (2025). *Energy and AI* (World Energy Outlook special report). https://www.iea.org/reports/energy-and-ai

ISO/IEC. (2022). *Artificial intelligence — Concepts and terminology* (ISO/IEC 22989:2022). International Organization for Standardization. https://www.iso.org/standard/74296.html

ISO/IEC. (2023a). *Artificial intelligence — Guidance on risk management* (ISO/IEC 23894:2023). International Organization for Standardization. https://www.iso.org/standard/77304.html

ISO/IEC. (2023b). *Artificial intelligence — Management system* (ISO/IEC 42001:2023). International Organization for Standardization. https://www.iso.org/standard/81230.html

ISO/IEC. (2025a). *Artificial intelligence — AI system impact assessment* (ISO/IEC 42005:2025). International Organization for Standardization. https://www.iso.org/standard/42005.html

ISO/IEC. (2025b). *Artificial intelligence — Requirements for bodies providing audit and certification of artificial intelligence management systems* (ISO/IEC 42006:2025). International Organization for Standardization. https://www.iso.org/standard/42006.html

Jakesch, M., Bhat, A., Buschek, D., Zalmanson, L., & Naaman, M. (2023). *Co-writing with opinionated language models affects users' views*. Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2302.00560

Johnson Spink, D. (2026). *The AI jester: How AI makes you confident and wrong*. LinkedIn. https://www.linkedin.com/pulse/ai-jester-how-makes-you-confident-wrong-johnson-spink-gg3df/

Khan, R., Joyce, D., & Habiba, M. (2025). *AGENTSAFE: A unified framework for ethical assurance and governance in agentic AI*. arXiv. https://arxiv.org/abs/2512.03180

Kolt, N. (2025). *Governing AI agents*. Notre Dame Law Review, 101 (forthcoming). https://arxiv.org/abs/2501.07913

Li, X., Tramèr, F., Liang, P., & Hashimoto, T. (2022). *Large language models can be strong differentially private learners*. International Conference on Learning Representations (ICLR 2022). https://arxiv.org/abs/2110.05679

Madatha, P. (2026). *A deterministic control plane for LLM coding agents*. arXiv. https://arxiv.org/abs/2606.26924

Mata v. Avianca, Inc., 678 F. Supp. 3d 443 (S.D.N.Y. 2023). https://www.law.berkeley.edu/wp-content/uploads/archive/2025/12/Mata-v-Avianca-Inc.pdf

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). *Model cards for model reporting*. Proceedings of the Conference on Fairness, Accountability, and Transparency (FAT* '19), 220–229. https://arxiv.org/abs/1810.03993

Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Cooper, A. F., Ippolito, D., Choquette-Choo, C. A., Wallace, E., Tramèr, F., & Lee, K. (2023). *Scalable extraction of training data from (production) language models*. arXiv. https://arxiv.org/abs/2311.17035

National Institute of Standards and Technology. (2023). *AI risk management framework (AI RMF 1.0)*. https://www.nist.gov/itl/ai-risk-management-framework

National Institute of Standards and Technology. (2024). *Artificial intelligence risk management framework: Generative artificial intelligence profile* (NIST AI 600-1). https://doi.org/10.6028/NIST.AI.600-1

Office of the Governor of California. (2025). *Governor Newsom signs SB 53, advancing California's world-leading artificial intelligence industry*. https://www.gov.ca.gov/2025/09/29/governor-newsom-signs-sb-53-advancing-californias-world-leading-artificial-intelligence-industry/

OpenAI. (2025). *Preparedness framework* (Version 2). https://cdn.openai.com/pdf/18a02b5d-6b67-4cec-ab64-68cdfbddebcd/preparedness-framework-v2.pdf

OpenAI. (2026b). *Hugging Face model-evaluation security incident*. https://openai.com/index/hugging-face-model-evaluation-security-incident/

OWASP. (n.d.). *OWASP top 10 for large language model applications*. https://owasp.org/www-project-top-10-for-large-language-model-applications/

OWASP. (2025). *Agentic AI: Threats and mitigations* (Version 1.1). OWASP Gen AI Security Project, Agentic Security Initiative. https://genai.owasp.org/resource/agentic-ai-threats-and-mitigations/

Palo Alto Networks. (2026). *2026 identity security landscape*. https://www.paloaltonetworks.com/idira/identity-security-landscape-report

Phuong, M., Aitchison, M., Catt, E., Cogan, S., Kaskasoli, A., Krakovna, V., Lindner, D., & Rahtz, M., et al. (2024). *Evaluating frontier models for dangerous capabilities*. arXiv. https://arxiv.org/abs/2403.13793

Raji, I. D., Smart, A., White, R. N., Mitchell, M., Gebru, T., Hutchinson, B., Smith-Loud, J., Theron, D., & Barnes, P. (2020). *Closing the AI accountability gap: Defining an end-to-end framework for internal algorithmic auditing*. Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency, 33–44. https://arxiv.org/abs/2001.00973

Romasanta, A., Thomas, L. D. W., & Levina, N. (2026). *Researchers asked LLMs for strategic advice. They got "trendslop" in return*. Harvard Business Review. https://hbr.org/2026/03/researchers-asked-llms-for-strategic-advice-they-got-trendslop-in-return

Russo, D. (2026). *Govern the repository, not the agent: Ecosystem-level risk in AI-native software*. arXiv. https://arxiv.org/abs/2606.28235

Sofroniew, N., Kauvar, I., Saunders, W., Chen, A., et al. (2026). *Emotion concepts and their function in a large language model*. Transformer Circuits Thread. https://transformer-circuits.pub/2026/emotions/index.html

South, T., Marro, S., Hardjono, T., Mahari, R., Whitney, C. D., Greenwood, D., Chan, A., & Pentland, A. (2025). *Authenticated delegation and authorized AI agents*. arXiv. https://arxiv.org/abs/2501.09674

Stanford Institute for Human-Centered AI. (2026). *The AI index 2026 annual report*. Stanford University. https://hai.stanford.edu/ai-index/2026-ai-index-report

United States Congress. (2025). *No Adversarial AI Act* (H.R. 4142, 119th Congress). https://www.congress.gov/bill/119th-congress/house-bill/4142

Wang, Quan, Yu, Hu, & Tsang. (2025). *Reflection-driven control for trustworthy code agents*. arXiv. https://arxiv.org/abs/2512.21354

Wang, Y., Zhang, J., Cai, T., et al. (2026). *From agent traces to trust: A survey of evidence tracing and execution provenance in LLM agents*. arXiv. https://arxiv.org/abs/2606.04990

Wang, Z., Schiller, N., Li, H., Narayana, S. S., Nasr, M., Carlini, N., Qi, X., Wallace, E., et al. (2026). *ExploitGym: Can AI agents turn security vulnerabilities into real attacks?* arXiv. https://arxiv.org/abs/2605.11086

White House. (2025). *Ensuring a national policy framework for artificial intelligence* (Executive Order 14365). https://www.whitehouse.gov/presidential-actions/2025/12/eliminating-state-law-obstruction-of-national-artificial-intelligence-policy/

Willison, S. (2026). *OpenAI's accidental cyberattack against Hugging Face is science fiction that happened*. Simon Willison's Weblog. https://simonwillison.net/2026/Jul/22/openai-cyberattack/

6 Mastery & Forward Practice

Surprisingly, my consulting skills are more in demand now than ever — not despite AI, but because of it. AI has democratised the production and dissemination of knowledge.. What clients now pay for is knowing what work to do — framing the ask, judging the result, verifying final outputs — which is exactly what the last five chapters have been teaching.

A book ends where a 道 should: not at a destination but at a practice you keep. So this closing chapter does four things. It looks back along the way travelled, chapter by chapter. It asks what stays human as the models keep improving. It reads the trend lines on where AI is heading — for the technology, for work, and for the societies both sit inside. And it ends with the loop you never stop running, and the first steps to take when you put the book down.

6.1 The Way Travelled

A 道 ends by looking back along the path, so here is the whole argument once more, chapter by chapter, as one walk.

Chapter 1 laid a deliberately unglamorous foundation: a large language model predicts the next token. Everything else follows from that one mechanism. Predicting text well forces the model to learn the structure of the world the text describes, which is why it can genuinely reason. Nothing in the mechanism checks truth, so it hallucinates with perfect confidence. And because its strength tracks the density of its training data, competence is jagged — an Olympiad gold one minute, a misread clock face the next. The foundations of the method are then articulated: treat the model as a loop, not an oracle, and stay responsible for what is produced.

Chapter 2 took that loop to the desk. A prompt you would use again became a skill. Wrapping a skill in a self-check made it dependable, and the best of them became tools that other agents could call. Everything worth telling a model became Markdown, context and memory turned single sessions into work that compounds, and the confidence trap — fluent, agreeable, wrong — set the habit of checking before trusting. The chapter ended with a personal operating model: plays, preferences, and a daily loop, assembled one working part at a time.

Chapter 3 turned those parts into software. Three real projects, each begun from a sentence, showed a non-programmer deploying working systems. They also showed why the craft is *intent, context, expectations*: say what the system is for and how you will know it is right, then leave the how to the agent. Over-specify the how and you fight the tool's strength; refuse to own the result and you end up with slop that nobody vouched for. Verification settled at the boundary, and the work itself left the editor.

Chapter 4 scaled one agent to many and found the human at the centre of the fleet. The disciplines split three ways: what the human must bring (intent, judgement, accountability, independence of mind), what the agent system must provide (harness, context and memory, orchestration, loops, evaluation), and what the two together require (division of labour, delegation and review, presence, calibrated trust). Its inversion is the hinge of the whole book: the better the agents get, the more the human disciplines matter, because your judgement becomes the limiting factor.

Chapter 5 was the duty that follows. Responsibility runs in a chain from the labs that govern their models, through the laws now arriving around the world and the standards that make compliance checkable, into the enterprise, and finally to a person with a name. Capability can be delegated; responsibility cannot.

Table 6.1: The book in one line per chapter — each discipline and the sentence to carry from it.

Chapter	The discipline	The sentence to keep
1 · Foundations	A loop, not an oracle	It predicts; it does not know
2 · Productivity	Build the system around the model	Turn what you know into Markdown, one clear ask at a time
3 · Software	Intent, context, expectations	Say what and how you'll check it; leave the how to the agent
4 · Disciplines	Manage the fleet; keep the judgement	The better the agents, the more the human matters
5 · Responsibility	The duty behind the capability	Capability can be delegated; responsibility cannot

That is the recap, and it is the easy part of a retrospective. The harder part is deciding which of these disciplines will still matter in a few years — and that depends on what the models can and cannot take over.

6.2 What Stays Human

Chapter 1 observed that we all draw on the same handful of frontier models, and every release narrows the gaps between them. When everyone holds the same tool, the difference between professionals has to live somewhere the tool is not. Three bodies of evidence, gathered over the course of writing this book, say where: in your expertise, in your judgement about when to trust the machine, and in the thinking you keep doing for yourself.

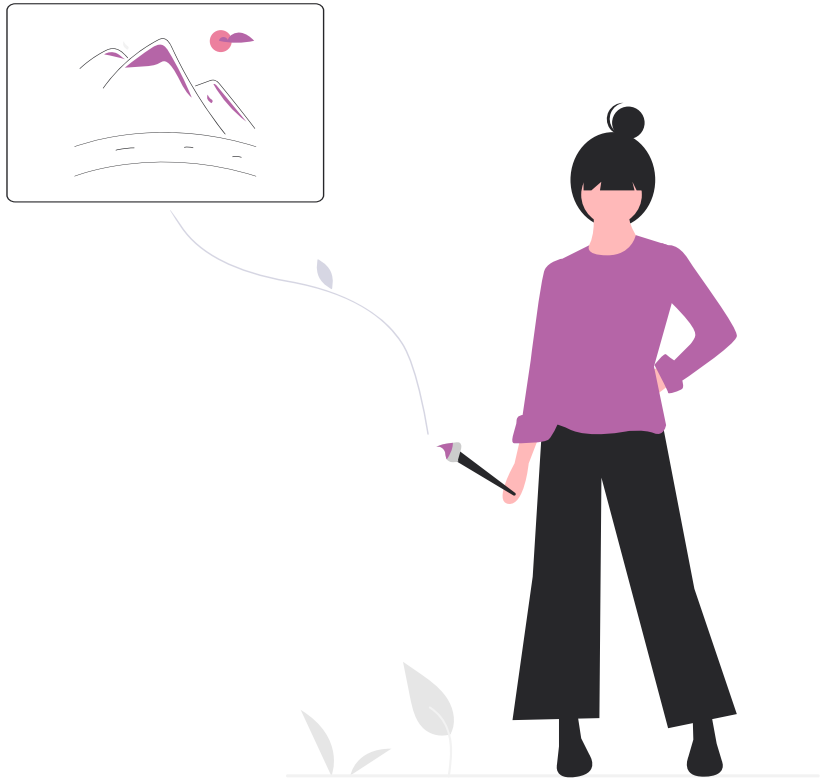


Figure 6.1: When everyone holds the same tool, the edge is the taste and expertise the tool cannot supply.

6.2.1 The Human Edge

As model quality converges, advantage moves to what cannot be downloaded: tacit expertise — the engineer or analyst who has sat with the real problem long enough to know why it matters and what "right" looks like there. The pattern in the data is augmentation, not replacement, and the firms that over-automated and then quietly rehired their seasoned staff make the point in reverse (Stanford HAI, *The AI index 2026 annual report*, 2026). There is a subtlety in who gains most, and it repays a careful reading. Measured productivity gains skew to novices: in one of the best-known field studies, an AI assistant lifted new customer-support agents far more than veterans, because the model had been trained on what the veterans already did — it packaged their expertise and handed it to beginners (Brynjolfsson et al., *Generative AI at work*, 2023). The expertise came first, and the tool only passed it along; the expertise is the thing worth having. And expertise now buys something it never could: the subject-matter expert who can say precisely what a system is for finally has a way to build it, as Chapter 3 showed — provided they keep enough judgement to steer.

The split between augmentation and automation is visible in wages, not just anecdotes. Studying US occupations from 2015 to 2022, Damien Marguerit separated AI that augments a job from AI that automates it, and found the two pull in opposite directions (Marguerit, *Augmenting or automating labor? The effect of AI development on new work, employment, and wages*, 2025). A one-standard-deviation rise in exposure to augmenting AI came with about 3.1% higher employment and higher wages for higher-skilled workers. The same rise in automating AI came with roughly 7.7% lower hourly wages for lower-skilled ones. The same technology, aimed differently, lifts one group and displaces another. Which side you land on is partly a choice: it depends on whether the tool is set up to extend your judgement or to replace your task.

You keep the edge by staying close to real problems and owning intent and judgement. You lose it — collectively — by hollowing out the junior pipeline that produces tomorrow's seniors. You save on this year's salaries, and then a decade on you find there is nobody ready to promote into the senior roles. That risk now has a measured shape. Across 11,097 open-source repositories, adopting coding agents left the number of human contributors flat, but cut the share of newcomers by 3.7 percentage points and deepened the review burden on maintainers by 5.3% — the authors call it *augmentation with dilution*: the entry rungs thinning while the load on seniors climbs (Zhang et al., *Augmentation with dilution: Human contributor ecosystems after AI coding agent adoption*, 2026). The measurement is in software because commits are easy to count, but the mechanism is general. Every field that grows its seniors

by handing juniors the work AI now absorbs faces the same thinning — junior analysts, associates, and research assistants alike.

There is an inner edge as well as an organisational one: your sense of your own judgement, and AI erodes it twice over. First, it lifts task scores while flattening metacognition, so strong and weak performers end up equally — and wrongly — sure of themselves; unsettlingly, the more someone knows about AI, the *worse* their self-assessment tends to become (Fernandes et al., *AI makes you smarter but none the wiser: The disconnect between performance and metacognition*, 2026). Second, your confidence drifts to match whatever confidence the model projects, and the pull lingers even after the model is gone (J. Li et al., *As confidence aligns: Effect of AI confidence on human self-confidence in human–AI decision making*, 2025). Protecting the human edge therefore means protecting your calibration. Form a view of your own before you ask the model, and treat its certainty as one more input to weigh rather than a verdict to accept.

The book's hand drawn illustrations are an affirmation of the above choice. They are the work of a human illustrator, Katerina Limpitsouni. I picked them from her open-source set and recoloured them to fit these pages, rather than asking a model to generate one for each section. Deciding which picture belongs beside which argument — and whether it earns a place at all — is a matter of taste, and taste is the human part this chapter keeps coming back to. So I chose them by hand, on purpose.

6.2.2 When the Pair Works

It is tempting to assume that a person and an AI together must beat either alone. The evidence says otherwise, and the correction matters. Michelle Vaccaro, Abdullah Almaatouq, and Thomas Malone pooled 106 experiments — 370 effect sizes — and found that, on average, human–AI teams performed *worse* than the better of the human alone or the AI alone: a small but real negative effect, a Hedges' g of −0.23 (Vaccaro et al., *When combinations of humans and AI are useful: A systematic review and meta-analysis*, 2024). The losses clustered in decision-making tasks; the gains, where they appeared, were in content creation. Synergy, in other words, does not happen by itself; somebody has to design the pairing for it.

The pattern underneath tells you when to expect it. Combinations helped when the human alone was the stronger of the two, and hurt when the AI alone was stronger — because in that second case the human's remaining job is to know when to defer, and people are bad at it. Jingshu Li and colleagues showed why. When an AI reports how confident it is, users mostly cannot tell whether that confidence is earned: only about a quarter noticed when a model was miscalibrated, against

three-quarters who could read a well-calibrated one (J. Li et al., *Understanding the effects of miscalibrated AI confidence on user trust, reliance, and decision efficacy*, 2024). An overconfident model pushed people into over-reliance; an underconfident one made them ignore good advice; and simply disclosing the miscalibration did not repair the decisions.

So the edge is calibration on both sides of the pairing. It works when the human can judge whether to trust a given answer, which means keeping enough independent expertise to check the machine rather than defer to it. This is the §6.2.1 warning about your own judgement, raised to the level of the team: how confident the model sounds tells you something about the model, and nothing about the world. Build the pairing so the person works on the parts they are better at and checks the parts the model is better at, always remembering that a confident answer and a correct one are two different things.

6.2.3 Keeping Your Own Mind

There is a cost to offloading thinking that never shows up on the finished page. Nataliya Kosmyna and colleagues at MIT had people write essays in three conditions — with an LLM, with a search engine, and unaided — while recording their brain activity. The unaided writers showed the strongest, most distributed neural connectivity; the search users less; the LLM users least (Kosmyna et al., *Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task*, 2025). More telling was memory and ownership: over 83% of the LLM writers could not quote a single sentence from the essay they had just produced, and asked later to write unaided, they stayed disengaged. The authors call the effect *cognitive debt* — fluency now, borrowed against understanding later. It is a small, preliminary study, and a formal commentary argues it is underpowered and that its brain-connectivity measure cannot carry the causal story (Stanković et al., *Commentary on Kosmyna et al. (2025), "Your brain on ChatGPT"*, 2026) — so treat the exact numbers as provisional; the direction is what matters.

The pattern shows up at population scale too. Michael Gerlich surveyed 666 people and found frequent AI-tool use correlated with lower critical-thinking scores, an effect statistically carried by cognitive offloading — handing the mental work to the tool (Gerlich, *AI tools in society: Impacts on cognitive offloading and the future of critical thinking*, 2025). It was strongest in the youngest users and softened by more education. Both studies are correlational, and neither proves the tool makes you worse. But the mechanism is plausible and worth guarding against: a muscle you stop using weakens.

The antidote is not to refuse the tool but to keep practising the very thing it can do for you. Decades before any of this, Anders Ericsson

and colleagues showed that expert performance is built by *deliberate practice* — effortful, feedback-driven repetition of exactly the hard part — not by talent or time served; their top violinists had accumulated on the order of ten thousand hours of it (Ericsson et al., *The role of deliberate practice in the acquisition of expert performance*, 1993). AI removes exactly the effortful struggle that deliberate practice depends on. Let it take every hard step and you stop building the judgement that lets you supervise it. So keep some of the hard part for yourself: write the first draft, work the problem before you ask, and reserve the reps that grow the skill you most want to keep.

6.3 Where AI Is Heading

Every chapter so far has described the tools as they are. This section reads the trend lines, because the measurements are now good enough to be worth reading, and the disagreements among the experts are themselves worth understanding.

Start with the clearest line we have. Researchers at METR asked a simple question: how long a task — measured by how long it takes a skilled human — can a model finish reliably? They call it the *time horizon*: the task length a model completes about half the time. Plotted against the calendar, it makes a straight line on a log scale. The horizon has been doubling roughly every seven months since 2019, from tasks that took a human seconds, to minutes, to the better part of an afternoon — their strongest model managed tasks around the two-hour mark (Kwa et al., *Measuring AI ability to complete long software tasks*, 2025). If the line holds — a real if, and the authors are careful about it — the horizon reaches a full working day within a few years. It is the most concrete way anyone has found to say what "the models are getting better" actually means.

There is fuel for a few more doublings. Epoch AI, an institute that tracks the inputs to AI rather than selling its outputs, examined whether training can keep scaling and concluded that runs about ten thousand times larger than GPT-4 are probably feasible by 2030 — the same size of leap again as GPT-2 to GPT-4 (Epoch AI, *Can AI scaling continue through 2030?*, 2024). What would stop it, if anything does, is physics and supply chains: electrical power first, then chip manufacturing, then training data, then hardware speed, in that order. There is fuel left, in other words — but for the first time you can see the gauge.

Whether more scale delivers *general* intelligence is exactly where the people who build these systems stop agreeing. A survey of 2,778 published AI researchers put the year they judged "high-level machine intelligence" — machines outperforming humans at every task — more

likely than not at around 2047, thirteen years sooner than the same community had guessed only a year earlier; and between a third and half of respondents gave at least a one-in-ten chance to an outcome as bad as human extinction (Grace et al., *Thousands of AI authors on the future of AI*, 2024). Yet when the field's main professional society convened its own panel in 2025, 76% of the researchers it surveyed said that simply scaling up today's methods was "unlikely" or "very unlikely" to get there at all (AAAI, *Presidential panel on the future of AI research*, 2025). Hold those two results side by side and the message is that there is no consensus timeline: the experts are all over the map, and anyone quoting a single confident date is selling something.

It also helps to stop treating "AGI" as a line the field crosses on some particular Tuesday. A team at Google DeepMind proposed a more useful frame: levels of AGI, graded on two axes — how *general* the ability and how *deep* — with today's best models placed on the bottom rung, "emerging AGI": as good as or somewhat better than an unskilled human across many tasks, but short of the next level, matching a *skilled* adult on most work (Morris et al., *Levels of AGI for operationalizing progress on the path to AGI*, 2023). On that map the interesting question stops being "is it AGI yet" and becomes "which cell is it in, and which tasks just moved."

The honest bookends are worth seeing side by side, because they are the two futures the argument runs between. Dario Amodei — who, as chief executive of Anthropic, has every reason to paint the ceiling high — sketches a "country of geniuses in a datacenter" compressing fifty to a hundred years of scientific progress into five or ten: most cancers cured, the human lifespan doubled (Amodei, *Machines of loving grace*, 2024). Against him, the Princeton computer scientists Arvind Narayanan and Sayash Kapoor argue for treating AI as *normal technology* — as transformative as electricity or the internet, and just as slow to land, because what gates impact is not raw capability but the plodding pace at which organisations and institutions adopt anything (Narayanan & Kapoor, *AI as normal technology*, 2025). Their historical anchor is worth keeping: the electric motor was available for some forty years before it showed up in productivity statistics, because factories had to be rebuilt around it first. Capability is not the same thing as impact, and the gap between the two is measured in how long institutions take to catch up — decades, on the dynamo's precedent. Nobody knows which bookend this ends nearer to — and the way of working this book teaches survives either way, because it invests in judgement rather than in a bet on the timeline.

6.4 The Future of Work

The human edge of §6.2 was one person's answer. This section asks the same question for the labour market as a whole — which jobs are touched, what people actually do with the tools, what the early employment data shows, and how big the whole thing really is.

Figure 6.2: The wave lands hardest on white-collar work — the jobs this book is written for.

Which jobs does this touch? The most-cited attempt to answer put human raters and GPT-4 side by side, scoring, for each occupation, what share of its tasks an LLM could do at least twice as fast without losing quality. The headline: around 80% of US workers have at least 10% of their tasks exposed, and about 19% have more than half exposed (Eloundou et al., *GPTs are GPTs: An early look at the labor market impact*

potential of large language models, 2024). The twist that makes this wave different from the last is *who* is exposed. Past automation came for routine manual and clerical work; this time exposure rises with wage and education. The knowledge workers who felt safest are the most exposed — which is precisely why this book is aimed at them.

That map of exposure is drawn for the United States, and it looks different across the world. Charting AI exposure across 141 economies, researchers found richer countries far more exposed than poorer ones — the white-collar jobs AI touches make up more of a wealthy economy — and women more exposed than men in nine countries out of ten. Even a country with little direct exposure is not insulated: some lean heavily on remittances from workers abroad whose jobs are exposed, so the shock arrives second-hand (Murugan et al., *The jagged global economy: Frontier AI unevenly exposes national economies*, 2026). Policy built for the US or European labour market, the authors warn, will not fit the rest of the world.

Exposure is potential, not fate, so it matters what people actually do with these tools. Anthropic, mining millions of conversations with its own Claude models, reported that real usage tilts towards *augmentation* over *automation* — roughly 57% of activity was a person working with the model rather than handing a task off wholesale — and clusters heavily in software and writing (Anthropic, *The Anthropic economic index*, 2025f). Treat the exact split gently: it is one vendor measuring its own product, and coders are over-represented among early adopters. But the direction matches everything else in this book — most useful AI work today involves a human staying in the loop rather than handing the task off entirely.

The first real employment data has arrived, and it lands on the young. Studying payroll records for millions of US workers, Erik Brynjolfsson and colleagues found that since generative AI spread in late 2022, employment for early-career workers aged 22 to 25 in the most AI-exposed occupations fell about 16% relative to everyone else — while older workers in the same jobs, and workers in less-exposed jobs, held steady or grew (Brynjolfsson, Chandar & Chen, *Canaries in the coal mine? Six facts about the recent employment effects of artificial intelligence*, 2025). It is the thinning of the entry rungs that §6.2.1 warned about, now visible in national data: the tasks a junior once cut their teeth on are the tasks AI absorbs first. The title's metaphor is apt and uneasy — the young in exposed fields are the canaries.

A set of interviews with software engineers puts a mechanism under that number. As AI takes over the code generation, debugging, and documentation that juniors once cut their teeth on, that work is quietly redirected into senior-plus-AI workflows and stops reaching the junior

at all — the researchers call it *absorption*. What a junior loses is not only the tasks but the *productive struggle* they carried, the difficulty that turns a beginner into an expert over years. One senior, asked where this leads, put the worry in a single question: "Who is going to become the next senior?" (Yu & Moon, *Who will become the next senior? How generative AI erodes the development pathway in software engineering*, 2026). It is a small interview study, so read it as a warning rather than proof — but the mechanism is easy to believe, and it is the thinning of the entry rungs seen from inside the team.

So is this a catastrophe or a rounding error? Start at the sober end of the range. The economist Daron Acemoglu, running the same task-exposure logic through a macroeconomic model, estimates AI will lift total factor productivity — the extra output squeezed from the same labour and capital — by *at most* about 0.66% over ten years, a fraction of the 7% GDP boom some banks forecast (Acemoglu, *The simple macroeconomics of AI*, 2024). His reason is simple once said: being *exposed* to AI and being *cheaply automatable* by it are different things, and only a modest slice of tasks clears both bars any time soon. The World Economic Forum, surveying more than a thousand employers about what they *expect*, tells a busier story — a churn of 22% of jobs by 2030, 170 million roles created and 92 million displaced, a net gain of 78 million (World Economic Forum, *Future of jobs report 2025*, 2025) — though that is what bosses predict, not what has happened, and predictions of this kind have a patchy record. The forecasts disagree so wildly because they measure different things. Task exposure, real usage, actual employment, and dollar output are four separate questions that keep getting reported under one headline:

Table 6.2: The four questions the "future of work" forecasts each answer, and what each found.

The question	Source	What it found
Task exposure	Eloundou et al., 2024	~80% of workers have at least 10% of tasks exposed; ~19% over half
Real usage	Anthropic, 2025f	~57% of activity is augmentation, not wholesale automation
Actual employment	Brynjolfsson, Chandar & Chen, 2025	early-career workers in exposed jobs down ~16% since 2022
Dollar output	Acemoglu, 2024	total factor productivity up *at most* ~0.66% over a decade

There is an optimistic reading worth taking seriously, precisely because its author refuses to call it a forecast. David Autor argues that AI could — if steered well — help *rebuild* the middle class that earlier automation hollowed out, by extending expert judgement to workers without elite credentials: the nurse practitioner safely doing more of what once required a doctor is his anchor example (Autor, *Applying AI to rebuild middle class jobs*, 2024). He is explicit that this is a possibility contingent on choices, not a property of the technology. Which returns the question to where §6.2 left it: whether AI augments you or replaces you depends partly on how the tool is set up, and partly on whether you keep the judgement this chapter has been describing. If the labour market keeps shifting, the augmented side of that line is the safer place to stand.

6.5 AI and Society

Work is where AI touches most professionals first, but not where its evolution matters most. Step back far enough and the questions change: what fluent machines do to a society's information and trust, what they do to learning, and who captures the gains.

Begin with a gap in perception, because it colours everything else. When Pew surveyed both the US public and AI experts, 56% of the experts expected AI to have a positive effect on the country over the next twenty years — against just 17% of the public (Pew Research Center, *How the US public and AI experts view artificial intelligence*, 2025). The

public is more worried than excited and expects job losses; the experts are the reverse. Strikingly, the two groups agree on the governance: both want more control over how AI is used in their lives, and both doubt the government will regulate it well. A technology arriving this fast and trusted this unevenly is a governance problem before it is a technical one — which is Chapter 5′s argument, seen from the street.

Part of the public's unease is well-founded, because these systems are quietly good at changing minds. In a controlled debate experiment, GPT-4 given a few facts about its opponent — age, politics, education — was markedly better at shifting views than a human opponent with the same information, raising the odds of moving someone by more than 80% (Salvi et al., *On the conversational persuasiveness of large language models*, 2025). The unsettling detail is that it worked even when people correctly guessed they were arguing with a machine. A tool that out-persuades humans at no cost and infinite scale, personalised to each target, is a genuinely new thing in the information environment — the same fluency that flatters your draft can be pointed at an electorate.

The subtlest cost lands on learning, where the struggle a tool removes is the very thing that was doing the teaching. In a trial with about a thousand high-school maths students, those given unrestricted GPT-4 to practise with did far better *during* practice — and then scored 17% *worse* than the no-AI group on an exam where the AI was taken away (Bastani et al., *Generative AI without guardrails can harm learning*, 2025). The tool had become a crutch that walked for them. The crucial finding is what fixed it: a version rebuilt as a tutor — hints, not answers — erased the harm entirely. Whether the tool taught or harmed came down to the guardrails, not the access. It is the lesson §6.2.3 drew from brain activity, repeated with exam scores and, this time, a remedy attached.

A 2026 experiment shows what decides the outcome: *how* the tool is used. Undergraduates given AI for a study task scored higher on a test straight afterwards, and — unlike the maths students — most of that gain was still there a week later when they worked without it. But the outcome split by habit. Students who used AI to *explain* concepts kept their gains; those who used it to *generate* the work outright saw the advantage disappear once the tool was taken away (Contractor & Reyes, *Experimental evidence on the learning impact of generative AI*, 2026).

The thread running through all of it is that AI is a lever, and a lever multiplies whatever force you already bring. Which way it tips is genuinely contested, and the disagreement is worth sitting with. When a task is well-defined and the tool can bottle an expert's know-how, AI lifts the least-skilled the most. In a study of more than five thousand customer-support agents, an AI assistant raised resolved cases by about

14% on average. But the gain was lopsided: roughly 34% for the newest, lowest-skilled staff, and barely any for the veterans, as if the experience the best staff had spent years building could be handed straight to a newcomer (Brynjolfsson et al., 2023). The writing experiment from §2.8 found the same levelling, the weakest writers gaining most (Noy & Zhang, 2023).

Move to open-ended work that turns on judgement, though, and it tips the other way. A field experiment with Kenyan entrepreneurs made this stark: an AI business mentor helped the already-strong performers by around 15% while leaving the weakest about 8% *worse* off (Otis et al., *The uneven impact of generative AI on entrepreneurial performance*, 2023). The gap did not come from the advice — both groups received good counsel — but from which advice each acted on: the strong performers could tell the useful from the useless. A study of tens of thousands of chess players is just as sobering. Once you account for who chooses to use the AI in the first place, its apparent teaching effect disappears, and access to it pushes the strong and the weak further apart (Riedl & Bogert, *Who benefits from AI? Self-selection, skill gap, and the hidden costs of AI feedback*, 2024).

What decides which way it tips? The evidence keeps pointing at the same thing, and it is what this book has been about all along: whether the person can direct and judge the tool. When researchers looked for what predicted who gained from AI on knowledge work, the best predictor was neither grades nor years of training. It was a distinct skill: how well someone could prompt the model, filter its answers, and check its work. A structured workflow, which supplies that discipline from the outside, sharply narrowed the spread of results (Idan & Anand, *Generative AI and the productivity divide*, 2026). That is the jagged frontier of §2.8 seen at the scale of a society (Dell'Acqua et al., 2023). AI closes the gap when the task itself supplies the skill a person lacks. When the skill has to come from the worker, it widens the gap instead.

The trouble is that this skill is unevenly held, and hardest to get for the people who most need it. Training in how to work with AI turns out to demand more prior schooling than ordinary training does, so it flows to those already well-equipped and passes over the workers most exposed to displacement (OECD, *Bridging the AI skills gap: Is training keeping up?*, 2025). Look from the workers' side and the story rhymes: among those most exposed to AI, the lower-paid meet it as a threat without the digital access, the training, or the slack to turn it to their advantage (Federal Reserve Bank of San Francisco, *On-the-job exposure to AI among lower-income workers*, 2025). Set beside §6.4's finding that this wave of exposure climbs the ladder of wage and education, the

picture is uncomfortable: the people who gain from AI and the people it hurts may not be the same people at all.

For those on the wrong side of that divide, AI is not a tool they pick up and use. It is a tool that puts them to work. The writer and activist Cory Doctorow has a sharp name for the reversal, the *reverse-centaur*. A centaur, in the older usage, is a person helped by a machine, with the person in charge. A reverse-centaur flips that: it is "a machine that uses a human being as its assistant" (Doctorow, *Reverse centaurs are the answer to the AI paradox*, 2025) — the worker kept to the machine's pace, left to take the blame when it errs. Whether AI lifts you or grinds you down, he argues, comes down to whether you chose it or it was imposed on you (Doctorow, *The Reverse Centaur's guide to life after AI*, 2026). That is the same line the evidence keeps drawing, now put as a question of power rather than skill. He writes as an activist making a case, not a researcher measuring one, so read him for the framing rather than the proof — but the framing is worth holding.

This is where a real social risk sits. A group that meets AI mostly as a threat — pushed out of a specialised job, handed the tool without the standing or skill to profit from it — may come to resent not just the outcome but the technology itself. The split in how people already feel is plain: in a 2025 US poll, 60% of the highest earners said AI does more good than harm, while a majority of the lowest earners said the reverse (Quinnipiac University Poll, 2025). How far that hardens into lasting hostility no one yet knows, and the honest evidence is thin. But there is a precedent worth heeding. Across thirteen European countries, people individually more exposed to an earlier wave of automation, the spread of industrial robots, became measurably more likely to vote for the radical right and less satisfied with democracy itself (Anelli, Colantone & Stanig, 2021). That study was about industrial robots and how people voted, not chatbots and how people feel, so take it as a warning rather than a forecast — and temper it with the small comfort that people tend to fear for other people's jobs more than their own. So the section ends where it began, only with more at stake: the aim is to be among the people who can judge what AI hands back, and to help build the training, safeguards, and institutions that bring more people to that point rather than leave them behind. Building those is the work of Chapter 5.

6.6 A Snapshot of the Field

The last sections read the evidence; this one reads the room. Another way to see where things are going is to watch where the people building these systems put their attention. In the middle of 2026 the largest gath-

ering of the practitioner community, the AI Engineer World's Fair, built its keynote programme around a single idea — the *software factory*, the autonomous, agent-run software pipeline Chapter 4 examined in §4.2 — with sessions from Microsoft, OpenAI, Hugging Face, and the open-weight labs behind models like GLM and MiniMax (AI Engineer, *Software factories and keynotes*, 2026). A conference bill is a snapshot of what an enthusiastic, self-selected crowd is excited about, not a measure of what works. Even so, it tells you what these engineers now spend their days on: fewer of them are perfecting a single prompt, and more are trying to keep a fleet of agents running.

One idea from the opening is worth carrying, because it names something this book keeps circling. Swix (Shawn Wang), who runs the conference, called the craft *loop-stacking*: the core skill, he argued, is knowing which loop you are working in — a single prompt, a self-checking task, an agent, a fleet — and choosing when to step up to the next one. He told it as a climb: in the beginning there was the token, then the chat, then tools, then goals, and now the automations and factories that stack loops on top of loops. That is the same ladder Chapter 1 drew from chatbot to agent ecosystem, seen from the floor of a very loud room, and it is the loop this chapter closes on. Like the rest of this snapshot, it is a practitioner's enthusiasm rather than a tested claim — worth hearing for the vantage point it gives, not for any proof it offers.

6.7 Continuous Refinement

The book closes on the idea it opened with: a path. Treat your practice as a loop — experiment, get feedback, refine, repeat — and keep AI human-centred at each turn. The evidence is consistent that the gain comes not from the tool but from redesigning work around it, which is why the high performers move pilots into production while others count demos (McKinsey & Company, *The state of AI*, 2025). It is the same finding Narayanan and Kapoor scaled up to history in §6.3: the factory had to be rebuilt around the electric motor before the motor paid.

The same loop scales up from a person to a whole company. Microsoft's chief executive, Satya Nadella, argues that a firm's real asset in this era is not the model it rents but the *learning loop* it builds on top — the workflows, evaluations, and accumulated judgement that improve with each use — because "you can offload a task, or even a job, but you can never offload your learning" (Nadella, *A frontier without an ecosystem is not stable*, 2026). His striking claim is that as a company's AI capability grows, its people become *more* valuable rather than less, since it takes human judgement to point the machine at the goals that matter — "without human direction, you have compute running in

circles.” Read it as an emerging industry view rather than a settled one — Nadella runs a company that sells the tools he is describing — but the shape of his argument is the book’s own, drawn one level larger. The model is the part a company can buy from anyone. What it builds around the model — the way its people and its systems get better together with each project — is the part a rival cannot copy.

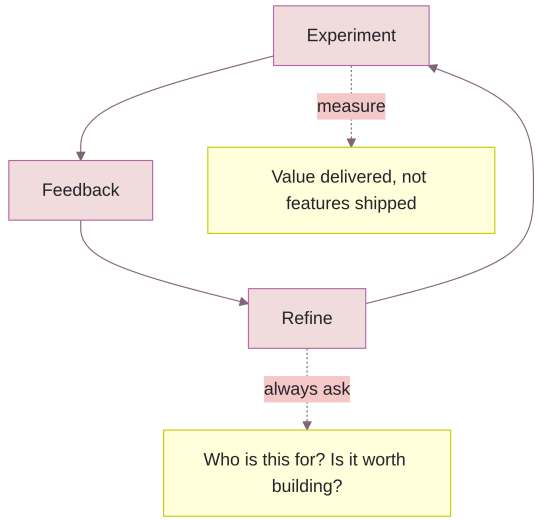


Diagram 6.1: The refinement loop — experiment, feedback, refine — with the two questions that keep it honest.

No method will be the last word, and that is the point of treating practice as a path rather than a destination. Vibe coding was the style of one year and looked spent within six months; spec-driven development began to buckle inside a year. The pattern is older and wider than software: business process re-engineering, Six Sigma, and a long line of agile relabellings were each sold as the last word, and each was quietly folded into whatever came next. What survives every relabelling is the discipline underneath — staying close to the work, holding the intent, asking what is worth building. That discipline is older than any of the frameworks. It is the Unix philosophy — do one thing well, compose small pieces, separate the *what* from the *how* — re-run on a tool that can now write the implementation itself, with a human kept at the centre (Chapter 1). So put your effort into the discipline underneath rather than into whichever framework is current this year. Build the skills that let you direct the machine, instead of racing it to produce more, and refuse to keep score by tokens burned — a number that says

how busy the machines were, not whether anything useful got made. One developer's single month ran to 603 billion tokens (Ahuja, *Spec-driven development is also breaking the fifty-year-old iron triangle*, 2026c; Ahuja, *Spec-driven development isn't broken. It will collapse*, 2026d).

Refinement has a collective hazard worth naming, because it echoes Chapter 5's convergence problem and §6.5's lever. Anil Doshi and Oliver Hauser had writers work with and without AI story ideas, and found a two-sided result (Doshi & Hauser, *Generative AI enhances individual creativity but reduces the collective diversity of novel content*, 2024). AI raised the novelty of an individual writer's story by 5 to 8%, and helped the least creative writers most. But the AI-assisted stories were measurably more like one another, drifting towards the same suggestions. Each writer was better off; the pool of stories was narrower. The defence is the one this book keeps returning to. Keep your own voice in the loop, and use the machine to sharpen a view you brought rather than to supply one. A tool that makes everyone sound alike carries a cost, however convenient it feels.

Measure value, not output. Counting features shipped feels good, and it quietly skips the question of whether anyone needed them. The way of AI turns out to be one question, asked again and again — who is this for, and is it worth building — until it hardens into habit. That habit, more than any model, is what AI-dō is for.

6.8 Shuhari — The Way From Here

A 道 has a shape to its learning, and the arts that end in 道 named it long before software did: *shuhari* — 守 *shu*, keep to the form; 破 *ha*, break from it; 離 *ri*, leave it behind and move freely (Endō, *Shu-ha-ri*, 1998). Read this book as its *shu*. Follow the forms closely at first — one clear ask at a time, intent kept apart from implementation, Markdown as the medium, verification at the boundaries, a human answerable for what is produced. As your judgement grows and the tools shift beneath you, enter *ha*: bend the forms, drop the ones that stop fitting, keep what holds. In time comes *ri*, where the forms are instinct and you invent your own — which is only the promise of Chapter 1 come round again: learn the philosophy, and the methods become yours to invent.

Figure 6.3: A 道 is a path you keep walking: shuhari is its shape — keep the form, break it, then move freely.

The rest is yours to begin, and the first steps are small and specific. First, take one real task you already do and run it as a loop — draft, check, refine — instead of waiting on a perfect prompt. Second, start an LLM wiki: put the context you keep re-explaining into a few Markdown notes the model can draw on. Third, add one verification step at the boundary that matters most, and name the person answerable for what leaves it. Fourth, before you open a chat window on anything that counts, write your own view down first, so you can tell the model's confidence from your own. Fifth, keep a second model on the bench, and ask of everything you make: who is this for, and is it worth building?

None of that rests on the tools. It rests on the method you build around them, on your judgement about which question is worth asking, and on the care you bring to the people your work touches. The tools will keep changing — that is the one safe prediction — and the way does not. Begin.

References

AAAI. (2025). *AAAI 2025 presidential panel on the future of AI research*. Association for the Advancement of Artificial Intelligence. https://aaai.org/wp-content/uploads/2025/03/AAAI-2025-PresPanel-Report-FINAL.pdf

Acemoglu, D. (2024). *The simple macroeconomics of AI* (NBER Working Paper No. 32487). National Bureau of Economic Research. https://www.nber.org/papers/w32487

Ahuja, K. V. (2026c). *Spec-driven development is also breaking the fifty-year-old iron triangle*. Activated Thinker (Medium). https://howtoarchitect.io/78431acba162?sk=cd2a36f452af96ccbfbcfcdeaa92ec06

Ahuja, K. V. (2026d). *Spec-driven development isn't broken. It will collapse*. Activated Thinker (Medium). https://howtoarchitect.io/c00609f72496?sk=2da01d7d2abfb5bc0acaed7050a0e797

AI Engineer. (2026). *WF2026: Software factories & keynotes ft. Microsoft, OpenAI, OpenClaw, Z.ai (GLM), MiniMax, HF* [Conference livestream]. YouTube. https://www.youtube.com/watch?v=htM02KMNZnk

Amodei, D. (2024). *Machines of loving grace: How AI could transform the world for the better*. https://www.darioamodei.com/essay/machines-of-loving-grace

Anelli, M., Colantone, I., & Stanig, P. (2021). *Individual vulnerability to industrial robot adoption increases support for the radical right*. Proceedings of the National Academy of Sciences, 118(47), e2111611118. https://doi.org/10.1073/pnas.2111611118

Anthropic. (2025f). *The Anthropic economic index*. https://www.anthropic.com/news/the-anthropic-economic-index

Autor, D. (2024). *Applying AI to rebuild middle class jobs* (NBER Working Paper No. 32140). National Bureau of Economic Research. https://www.nber.org/papers/w32140

Bastani, H., Bastani, O., Sungu, A., Ge, H., Kabakcı, Ö., & Mariman, R. (2025). *Generative AI without guardrails can harm learning: Evidence from high school mathematics*. Proceedings of the National Academy of Sciences, 122(26). https://doi.org/10.1073/pnas.2422633122

Brynjolfsson, E., Chandar, B., & Chen, R. (2025). *Canaries in the coal mine? Six facts about the recent employment effects of artificial intelligence*. Stanford Digital Economy Lab. https://digitaleconomy.stanford.edu/publications/canaries-in-the-coal-mine/

Brynjolfsson, E., Li, D., & Raymond, L. R. (2023). *Generative AI at work* (NBER Working Paper No. 31161). National Bureau of Economic Research. https://www.nber.org/papers/w31161

Contractor, Z., & Reyes, G. (2026). *Experimental evidence on the learning impact of generative AI*. arXiv. https://arxiv.org/abs/2607.08849

Dell'Acqua, F., McFowland III, E., Mollick, E. R., Lifshitz-Assaf, H., Kellogg, K., Rajendran, S., Krayer, L., Candelon, F., & Lakhani, K. R. (2023). *Navigating the jagged technological frontier: Field experimental evidence of the effects of artificial intelligence on knowledge worker productivity and quality* (Harvard Business School Working Paper No. 24-013). Harvard Business School. https://www.hbs.edu/faculty/Pages/item.aspx?num=64700

Doctorow, C. (2025). *Reverse centaurs are the answer to the AI paradox*. Pluralistic. https://pluralistic.net/2025/09/11/vulgar-thatcherism/

Doctorow, C. (2026). *The Reverse Centaur's guide to life after AI: How to think about artificial intelligence before it's too late*. Verso Books. https://www.versobooks.com/products/3584-the-reverse-centaur-s-guide-to-life-after-ai

Doshi, A. R., & Hauser, O. P. (2024). *Generative AI enhances individual creativity but reduces the collective diversity of novel content*. Science Advances, 10(28), eadn5290. https://doi.org/10.1126/sciadv.adn5290

Eloundou, T., Manning, S., Mishkin, P., & Rock, D. (2024). *GPTs are GPTs: An early look at the labor market impact potential of large language models*. Science, 384(6702), 1306–1308. https://arxiv.org/abs/2303.10130

Endō, S. (1998). *Shu-ha-ri* [Essay]. Aikidō Saku Dōjō. https://da2el.wordpress.com/2018/06/20/shu-ha-ri-endo-seishiro-aikido-saku-dojocho/

Epoch AI. (2024). *Can AI scaling continue through 2030?* https://epoch.ai/blog/can-ai-scaling-continue-through-2030

Ericsson, K. A., Krampe, R. T., & Tesch-Römer, C. (1993). *The role of deliberate practice in the acquisition of expert performance*. Psychological Review, 100(3), 363–406. https://doi.org/10.1037/0033-295X.100.3.363

Federal Reserve Bank of San Francisco. (2025). *On-the-job exposure to AI among lower-income workers* (Community Development Research Brief No. 2025-03). https://www.frbsf.org/research-and-insights/publications/community-development-research-briefs/2025/11/job-exposure-to-ai-among-lmi-workers/

Fernandes, D., et al. (2026). *AI makes you smarter but none the wiser: The disconnect between performance and metacognition*. Computers in Human Behavior, 168, 108779. https://doi.org/10.1016/j.chb.2025.108779

Gerlich, M. (2025). *AI tools in society: Impacts on cognitive offloading and the future of critical thinking*. Societies, 15(1), 6. https://doi.org/10.3390/soc15010006

Grace, K., Stewart, H., Sandkühler, J. F., Thomas, S., Weinstein-Raun, B., & Brauner, J. (2024). *Thousands of AI authors on the future of AI*. AI Impacts. https://arxiv.org/abs/2401.02843

Idan, L., & Anand, B. (2026). *Generative AI and the productivity divide: Human–AI complementarities in education and knowledge work*. arXiv. https://arxiv.org/abs/2605.18143

Kosmyna, N., Hauptmann, E., Yuan, Y. T., Situ, J., Liao, X.-H., Beresnitzky, A. V., Braunstein, I., & Maes, P. (2025). *Your brain on ChatGPT: Accumulation of cognitive debt when using an AI assistant for essay writing task*. arXiv. https://arxiv.org/abs/2506.08872

Kwa, T., West, B., Becker, J., Deng, A., Kinniment, M., Rush, N., et al. (2025). *Measuring AI ability to complete long software tasks*. METR. https://arxiv.org/abs/2503.14499

Li, J., et al. (2025). *As confidence aligns: Effect of AI confidence on human self-confidence in human–AI decision making*. Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. https://arxiv.org/abs/2501.12868

Li, J., Yang, Y., Zhang, R., Liao, Q. V., Song, T., Xu, Z., & Lee, Y.-C. (2024). *Understanding the effects of miscalibrated AI confidence on user trust, reliance, and decision efficacy*. arXiv. https://arxiv.org/abs/2402.07632

Marguerit, D. (2025). *Augmenting or automating labor? The effect of AI development on new work, employment, and wages*. arXiv. https://arxiv.org/abs/2503.19159

McKinsey & Company. (2025). *The state of AI*. https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai

Morris, M. R., Sohl-Dickstein, J., Fiedel, N., Warkentin, T., Dafoe, A., Faust, A., Farabet, C., & Legg, S. (2023). *Levels of AGI for operationalizing progress on the path to AGI*. https://arxiv.org/abs/2311.02462

Murugan, A., Aguirre, T., Nagaraj, A., & Bommasani, R. (2026). *The jagged global economy: Frontier AI unevenly exposes national economies*. arXiv. https://arxiv.org/abs/2607.05404

Nadella, S. (2026). *A frontier without an ecosystem is not stable*. X (formerly Twitter). https://x.com/satyanadella/status/2066182223213293753

Narayanan, A., & Kapoor, S. (2025). *AI as normal technology*. Knight First Amendment Institute, Columbia University. https://knightcolumbia.org/content/ai-as-normal-technology

Noy, S., & Zhang, W. (2023). *Experimental evidence on the productivity effects of generative artificial intelligence*. Science, 381(6654), 187–192. https://doi.org/10.1126/science.adh2586

OECD. (2025). *Bridging the AI skills gap: Is training keeping up?* OECD Publishing. https://www.oecd.org/en/publications/bridging-the-ai-skills-gap_66d0702e-en.html

Otis, N. G., Clarke, R., Delecourt, S., Holtz, D., & Koning, R. (2023). *The uneven impact of generative AI on entrepreneurial performance* (Working Paper No. 24-042). Harvard Business School. https://www.hbs.edu/faculty/Pages/item.aspx?num=65159

Pew Research Center. (2025). *How the U.S. public and AI experts view artificial intelligence*. https://www.pewresearch.org/internet/2025/04/03/how-the-us-public-and-ai-experts-view-artificial-intelligence/

Quinnipiac University Poll. (2025). *The age of artificial intelligence: Americans wary of impact on daily life, see harm to education, benefits to medical advances* (Release No. 3923). https://poll.qu.edu/poll-release?releaseid=3923

Riedl, C., & Bogert, E. (2024). *Who benefits from AI? Self-selection, skill gap, and the hidden costs of AI feedback*. arXiv. https://arxiv.org/abs/2409.18660

Salvi, F., Horta Ribeiro, M., Gallotti, R., & West, R. (2025). *On the conversational persuasiveness of large language models: A randomized controlled trial*. Nature Human Behaviour. https://arxiv.org/abs/2403.14380

Stanford Institute for Human-Centered AI. (2026). *The AI index 2026 annual report*. Stanford University. https://hai.stanford.edu/ai-index/2026-ai-index-report

Stanković, M., Hirche, E., Kollatzsch, S., & Doetsch, J. N. (2026). *Commentary on Kosmyna et al. (2025), "Your brain on ChatGPT"*. arXiv. https://arxiv.org/abs/2601.00856

Vaccaro, M., Almaatouq, A., & Malone, T. (2024). *When combinations of humans and AI are useful: A systematic review and meta-analysis*. Nature Human Behaviour, 8(12), 2293–2303. https://doi.org/10.1038/s41562-024-02024-1

World Economic Forum. (2025). *Future of jobs report 2025*. https://www.weforum.org/publications/the-future-of-jobs-report-2025/

Yu, S., & Moon, T. (2026). *Who will become the next senior? How generative AI erodes the development pathway in software engineering*. arXiv. https://arxiv.org/abs/2607.17067

Zhang, Jiang, & Koziolek. (2026). *Augmentation with dilution: Human contributor ecosystems after AI coding agent adoption*. arXiv. https://arxiv.org/abs/2606.26289

Changelog

Edition 1.1 — released 24 July 2026. - New in Chapter 5: the July 2026 OpenAI–Hugging Face model-evaluation cyberattack (§5.1.1, §5.5.1), plus measured evidence that reinforcement learning grows a model's reward-seeking. - New in Chapter 6 (§6.5): a fuller treatment of AI's uneven benefits — who it lifts and who it leaves behind, Cory Doctorow's *reverse-centaur*, and the risk of a displaced group turning against AI. - Updated Chapter 5 (§5.2): Australia's National AI Plan and Office of AI, and the rise of Chinese open-weight frontier models with the risk of a US ban. - New across Chapters 4–6: current July 2026 research on independence of mind, where developers draw the line on AI autonomy, calibrated trust, the global spread of AI exposure, the erosion of the path from junior to senior, and what makes AI help or harm learning. - New in Chapter 4 (§4.2.1): Lilian Weng on harness engineering as an emerging practitioner view. - Site: improved search-engine and social-share metadata.

Edition 1.0 — released 8 July 2026.

Index

F

G

H

I

J

L

M

N

O

P

R

S

T

V

W

Z

AI-dō

The Way of AI, grounded in practice (道)

Some AI guides and online articles focus on tools, prompts, and techniques. Those go stale in the next release. *AI 道* is a practical, opinionated guide to using artificial intelligence effectively.

Method over model — results come from how you specify intent, gather context, and verify the outcome, not from which model you pick.
Discipline over vibe — principles that transfer across tools and survive model releases, not one-off prompts.
Augmentation, not replacement — AI transforms and amplifies human judgement and creativity rather than replacing them.

Written by a consultant and educator, grounded throughout in primary sources and peer-reviewed research. Each chapter states the concept, then shows how it works with real examples. It is for thoughtful professionals (leaders, consultants, analysts, designers and builders). Read it as a practice guide, not a reference manual or textbook. With twelve beautiful illustrations by Katerina Limpitsouni ([unDraw](https://undraw.co)), recoloured to the book's palette.

Chris Tham · Hello Tham
Read online, or get the PDF and ePub: christham.net/aidou