ROD & STAFF MEDIA

THE FIELD GUIDE

The Field Guide to AI

The Complete Manual

How artificial intelligence actually works, and how to build with it — from the very first idea to the builder's craft. In plain language.

LIVING MANUAL · V1 AUTHOR ROD & STAFF MEDIA PARTS I–IV

GUIDING PEOPLE THROUGH NEW GROUND.

CONTENTS

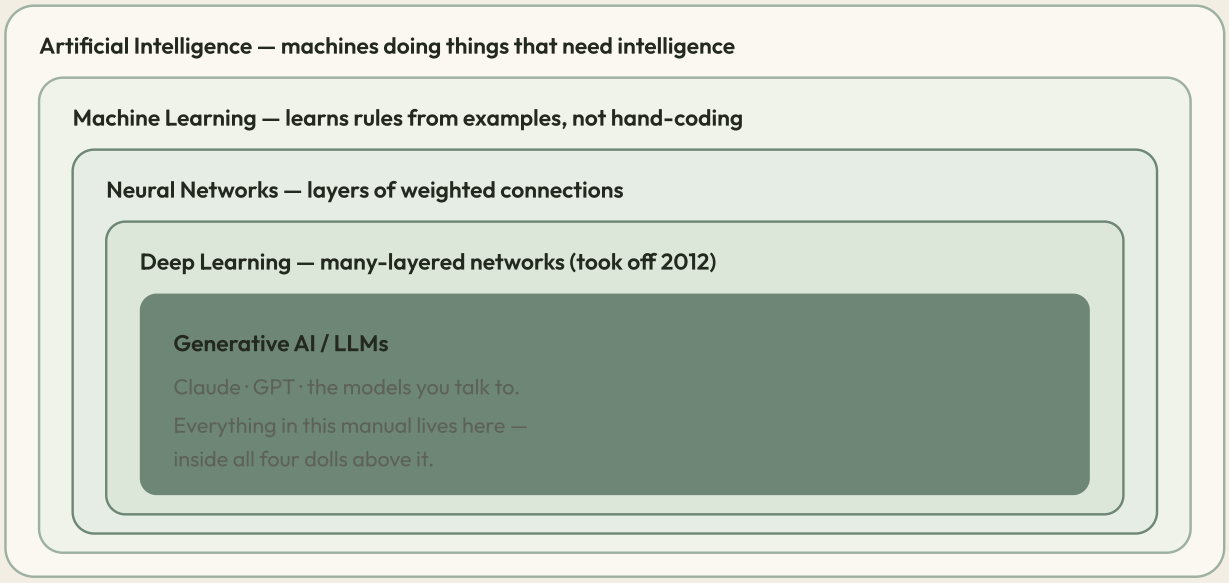
What's inside

HOW TO READ THIS

Start at the beginning. Each idea is built on the last.

This manual is the whole journey in one place — the same lessons, distilled into a reference you can keep. Read it in order: every part stands on the one before it, and the words in `mono` are defined in the Glossary at the back.

Before we start, one map. The word "AI" gets used for five different things, and they're nested inside each other like Russian dolls. When you use a tool like Claude, you're using all five at once — each one sitting inside the next.



THE NESTING DOLLS OF AI

PART ONE

Think—

How the machine actually thinks. Four ideas that, stacked together, explain every modern AI system — no math required.

1.1 Rules vs. Examples

> *For all of computing history, humans wrote the rules. Machine learning flipped it: show the machine examples, and it works out the rules itself.*

Traditional software runs on rules a human wrote by hand — *if this, then that*. A programmer anticipates every case and spells out exactly what to do. That's perfect for taxes and spreadsheets, and it falls apart the moment a problem gets fuzzy and human.

Take a simple task: *is there a dog in this photo?* Try to write that as rules. "Has fur" — so does a cat. "Four legs" — so does a table. You could write rules for a century and never capture every dog. Machine learning takes the opposite path: you show it ten thousand photos labelled "dog" and ten thousand labelled "not dog," and it learns the pattern itself — the way a toddler learns "dog" from being shown dogs, not handed a definition.

FROM WRITING RULES TO LEARNING FROM EXAMPLES

ANALOGY

A toddler learns "dog" by seeing dogs, never a dictionary definition. Same with the machine — pattern from examples, not rules from a programmer.

USE IT IN YOUR BUILD

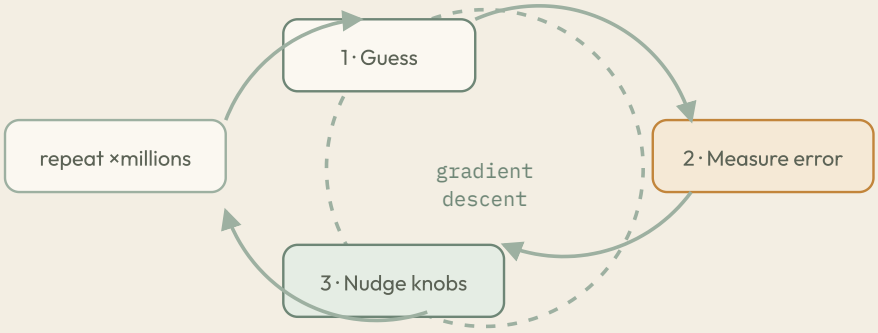
This is why AI can do the messy-human work old software never could: write, summarize, read a CRM, judge a tone. When a task is fuzzy, reach for learning-from-examples — not hand-written rules.

1.2 Inside the Network

> *A neural network is a wall of billions of little knobs. It learns by being wrong, over and over, and nudging each knob a hair toward being less wrong.*

Don't picture a brain — picture a giant mixing board with billions of knobs. Each knob is a `weight`. Information flows in one side, through all the knobs, and an answer comes out the other. At first the knobs are random, so the first answers are garbage. That's expected.

Then the magic: compare the answer to the right answer, measure how wrong it was (the `error`), and nudge every knob a hair in the direction that would have made it *less* wrong. That backward "assign blame, then adjust" step is `backpropagation`. Do it across millions of examples and the knobs settle into settings that just work. The slow walk toward less error is `gradient descent` — a hiker feeling downhill through fog, one step at a time.



HOW A NETWORK LEARNS — THE ERROR LOOP

ANALOGY

Tuning a guitar by ear: pluck, hear how far off it is, turn the peg a touch, repeat. The "knowledge" lives in the final knob positions — not in any fact written down.

KEY TERMS

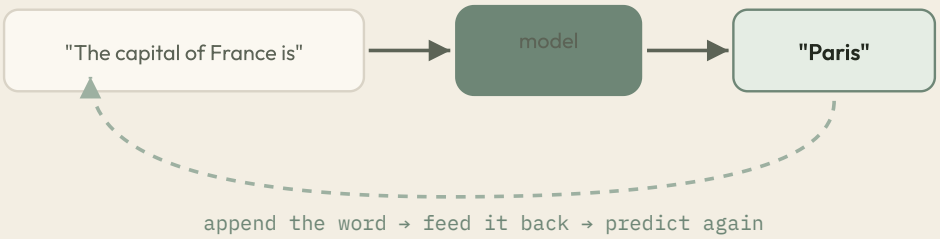
`weights`	the billions of tunable numbers; where the "knowledge" lives
`backpropagation`	working backward from a mistake to adjust every knob
`gradient descent`	stepping "downhill" toward less error, over and over

1.3 The Next Word

> *A language model does one thing: predict the next word. Run that in a loop and it writes anything — and to get good at it, it had to learn almost everything.*

Generation is recognition, run in a loop. The model predicts the single most likely next `token` (a word-piece), sticks it on the end, feeds the whole thing back into itself, and predicts again. One word, add it, ask again — a few hundred times — and out comes an email, a story, a block of code. It is never "writing a sentence." It is only ever predicting the very next word.

Here's the deep part. To predict the next word *well*, the model is forced to absorb grammar, facts, reasoning, and tone. "The capital of France is ___" requires geography. The intelligence is a side-effect of getting superhuman at one simple game. It works in probabilities; the `temperature` dial sets how adventurous its pick is — and that same machinery is exactly why it can be confidently, fluently wrong (a `hallucination`).



WRITING = PREDICTING THE NEXT WORD, IN A LOOP

ANALOGY

Phone autocomplete, scaled to a staggering degree. The difference is that getting *truly* good at autocomplete across the whole internet quietly requires understanding the world.

USE IT IN YOUR BUILD

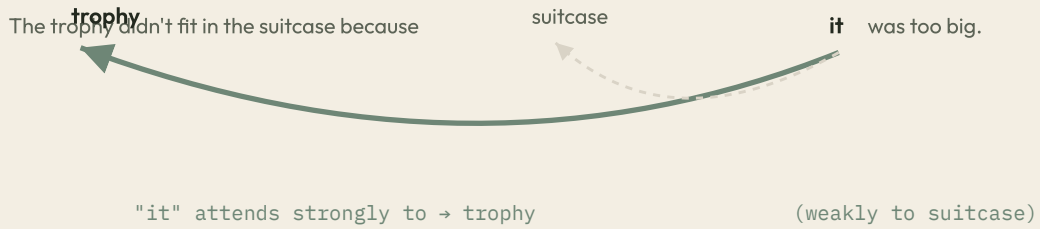
Because it can't *not* produce a next word, it will bluff when it doesn't know. Half of using AI well is knowing when to trust that fluency and when to verify it.

1.4 Attention & the Transformer

> *The 2017 idea that runs all of it: let every word look at every other word at once and decide what matters. That mechanism is the Transformer — the "T" in GPT.*

Before 2017, models read text in order, one word at a time, passing a little summary along — and by forty words deep, the start was a blur. The paper *"Attention Is All You Need"* threw that out for one idea: `self-attention`. Let every word look at every other word in the passage simultaneously and decide which ones are relevant.

In *"The trophy didn't fit in the suitcase because it was too big,"* attention is what lets "it" shine a spotlight across the sentence and land on "trophy," not "suitcase." Three things fell out of this and changed everything: it's **parallel** (train on GPUs at massive scale), it handles **long-range** connections, and it **scales** — bigger model + more data keeps getting smarter. That machine is the `Transformer`, sitting under Claude, GPT, and nearly every model you touch.



SELF-ATTENTION — EVERY WORD WEIGHS EVERY OTHER WORD

LANDMARK

2017	"Attention Is All You Need" (Vaswani et al., Google) introduces the Transformer
the "T"	GPT = Generative Pre-trained **Transformer**
scaling	more model + more data ⟶ reliably more capable; the engine of the modern boom

PART TWO

Wield—

How to make it work for you. Three tools that turn understanding into leverage — and into things people pay for.

2.1 PROMPTING · 2.2 GIVING IT KNOWLEDGE (RAG) · 2.3 AGENTS

2.1 Prompting

> *If the output is just the most likely continuation of the input, then whoever controls the input controls the output. Prompting isn't asking — it's setting the stage.*

Picture the model as the greatest improv actor alive — infinite range, but total amnesia. Every time the curtain rises it remembers nothing; all it has is the script you hand it, and it instantly becomes that character and plays the scene forward. Your prompt is the script. That's why two people get wildly different results from the same model — one hands it a sticky note, the other a full screenplay.

Four levers follow directly: **It's an amnesiac** — if it's not on the page, it doesn't exist, so give it the context. **Cast the role** ("you are a senior estimator…") to aim it at the expert region of what it learned. **Show, don't tell** — paste one example of "great" and it matches the pattern instantly. **Be specific** — vagueness gets you the beige, average answer, because that's the safest next word.

ANALOGY

A brilliant actor with amnesia. Hand him a thin note, you get generic improv. Hand him a rich script — who he is, the scene, one great line — and you get an Oscar performance.

USE IT IN YOUR BUILD

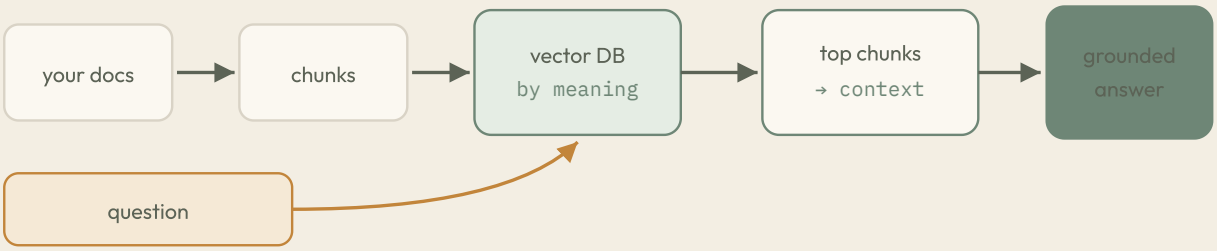
An assistant that "sounds like you" is just a well-written system prompt: role + context + tone + an example. You're not asking the model — you're casting it.

2.2 Giving It Knowledge — RAG

> *A model's knowledge is frozen, and it knows nothing private. RAG fixes that: turn its closed-book exam into an open-book one — slide the right page under its nose right before it answers.*

Without help, the model answers from memory and bluffs when it's stumped. RAG — retrieval-augmented generation — changes that. You chop your documents into `chunks`, store them in a `vector database` that files them by *meaning* (using `embeddings` — meaning as coordinates), and for each question you retrieve the few most relevant chunks and drop them into the context before the model answers.

Now the answer comes from real, current, private text — so the model reads instead of recalls, hallucination drops, and it can cite its source. And because the knowledge lives *outside* the model, you update it by editing a document, never retraining.



THE RAG PIPELINE — RETRIEVE THE RIGHT PAGE, THEN ANSWER

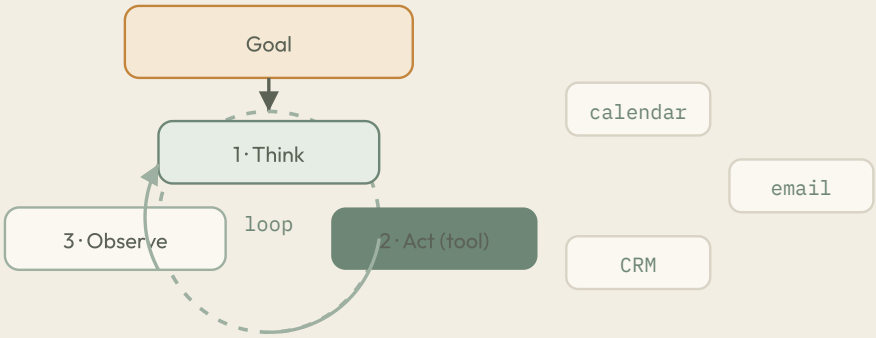
USE IT IN YOUR BUILD

A client chatbot that answers from *their* real pricing and policies — current, private, and honest — is RAG. It's a product you can build and charge for.

> *An agent is a brain, hands, and a loop. The model can only output text — so it outputs a request to use a tool; your program runs it for real and feeds the result back. Think, act, observe, repeat.*

A model alone can only talk. An `agent` gives it hands: `tools` — a calendar, a CRM, an email, a web search. The trick is simple: the model outputs a special piece of text meaning "use the calendar tool now"; the program wrapped around it actually runs the calendar, then pastes the result back into the model's context. The model reads that and decides its next move. That loop — think, act, observe — run until the goal is met, *is* the agent.

This turns a thing that *talks* into a thing that *does*: books the appointment, updates the record, sends the follow-up. The difference between a chatbot and a digital employee.



THE AGENT LOOP — THINK, ACT, OBSERVE, REPEAT

USE IT IN YOUR BUILD

Áine's morning briefing *is* this loop: think "I need the calendar" → call it → read it → "now the ad numbers" → call them → then write the summary. A brain, using hands, in a loop.

PART THREE

Go Deeper—

Make it yours, and stay ahead. Customizing models, knowing when they're good, and reading where the whole field is going.

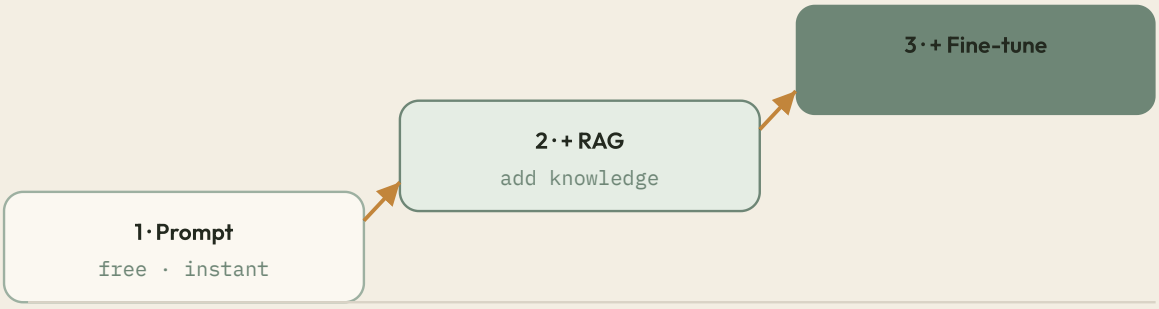
3.1 FINE-TUNING · 3.2 EVALS · 3.3 THE FRONTIER

3.1 Fine-Tuning

> *Fine-tuning is more training on your own examples. It changes the model's behavior and style — not its knowledge. Knowledge is RAG's job.*

A model is born in two rounds: **pretraining** on the whole internet makes a general word-predictor, then a lighter round of `fine-tuning` shapes it into a helpful assistant. You can run that second round yourself — take a trained model and nudge its knobs further on a focused set of example pairs, until it does a task in a style automatically.

The make-or-break distinction: fine-tuning changes *how it behaves*, not *what it knows*. Want it to know today's pricing? That's RAG — facts change, and baked-in facts go stale. Want it to write in your exact voice, or sort every lead the same way, ten thousand times? That's fine-tuning. The smart order is an escalation ladder: **prompt first** (free, instant), **add RAG** if it needs knowledge, and **fine-tune** only when you need rock-solid behavior at scale.



THE ESCALATION LADDER — PROMPT → RAG → FINE-TUNE

ANALOGY

Prompting is instructing a new hire for one task. RAG is handing them the company binder. Fine-tuning is the apprenticeship — practice until the skill is just in their hands.

USE IT IN YOUR BUILD

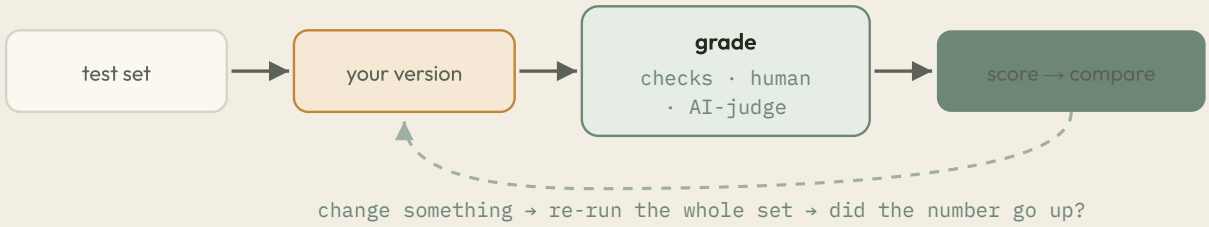
Estimates that always sound like a 15-year veteran → fine-tune the *behavior*. The current prices in them → RAG the *facts*. Two tools, two jobs.

3.2 Evals

> *An eval is a systematic taste test: a fixed set of cases plus a way to grade them — so you know whether a change actually helped, instead of guessing on vibes.*

The model is random (remember `temperature`), so the same prompt can shine on Tuesday and flop on Wednesday. Judging it by a single output is tasting one spoonful of soup and declaring the whole pot perfect. An `eval` fixes that: build a test set of representative cases, and every time you change anything — the prompt, the model, the retrieval — run the whole set and score it. Now you can compare versions with numbers.

You grade three ways: **hard checks** (did it output valid JSON? the right category?), **human rating** (the gold standard for taste, but slow), and **LLM-as-judge** (another model grading the first against a rubric — fast and scalable). And the mindset that makes it powerful: define what "good" looks like *before* you build, so you have a target to measure against.



AN EVAL — DEFINE "GOOD," THEN SCORE EVERY VERSION

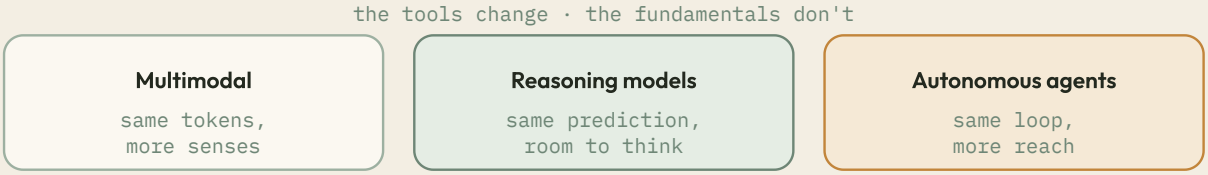
USE IT IN YOUR BUILD

"It answers correctly 98% of the time on our test set, and we'll know the moment that drops" — that sentence is the difference between a demo and a product someone pays for monthly.

> *Every "scary new breakthrough" is one of these same ideas, grown. The tools change every few months; the fundamentals barely move. That gap is your relevance moat.*

Three directions the field is moving — and each is something you already understand, with a new limb. **Multimodal:** models that see, hear, and speak — same tokens, same Transformer, just pointed at pixels and sound. **Reasoning models:** trained to think on a scratchpad before answering — still next-word prediction, just given room to reason. **More autonomous agents:** the same think-act-observe loop, stretched longer and trusted further — from tool you operate to teammate you delegate to.

The lesson under all of it: you didn't learn this year's vocabulary, you learned the *grammar* of the field. Every future headline reads as "that's just attention plus scale," or "a sharper eval," or "an agent with longer reach." And as the machines get better at producing, the rare human things — judgment, taste, trust, asking the right question — grow *more* valuable, not less.



THE FRONTIER — NEW LIMBS ON IDEAS YOU ALREADY KNOW

PART FOUR

The Builder's Craft—

Mastery. The hands-on disciplines that separate playing with AI from building things people pay for. (In progress — this part keeps growing.)

4.1 CONTEXT ENGINEERING · 4.2 THINK BEFORE YOU ANSWER · 4.3 OUTPUT SHAPING
4.4 RAG, DONE RIGHT · 4.5 AGENT DESIGN

4.1 Context Engineering

> *The model's working memory is the context window — a finite workbench. The real skill isn't clever wording; it's curating exactly what goes on that bench each call.*

Every time the model runs, the only thing it knows is the text in its `context window`, and from that it predicts the next word. So its entire behavior is decided by what's in the window. Most people think the skill is "writing a clever prompt." It isn't — it's deciding what to load into that window in the first place.

Picture the window as a workbench. Finite surface. Your job each call is to lay out exactly the right tools: who the model is, the specific facts it needs, the task, maybe one example. And here's the part that surprises people — **more is not better**. Dump your whole pantry on the bench and the model does *worse*: irrelevant clutter dilutes its attention. A tight brief beats a giant dump. Even placement matters — models attend most to the beginning and end, so the middle of a huge context can get lost.

THE CONTEXT WINDOW AS A WORKBENCH

ANALOGY

A chef's prep station. Lay out exactly the right ingredients and they cook beautifully. Pile the whole pantry on and they fumble for the salt.

4.2 Think Before You Answer

> *The model's only scratchpad is the text it writes. Make it reason step by step before the final answer, and hard problems get dramatically better.*

Recall that the model writes one word at a time, feeding its own output back in as it goes. That means it has no hidden place to think — its only thinking space is the text it actually produces. So if you demand *only* the answer to a hard question, you force it to blurt: commit to the first word instantly, with no room to work. Ask it to reason out loud first, and those steps become context that guides it to a far better conclusion. You're handing it paper.

This is the same engine behind reasoning models — but you can summon it for free, with one instruction: "think it through step by step," "list the key factors, then recommend." It leans on good context (4.1) and it cuts hallucination, because the model has to *earn* the conclusion instead of leaping to it. The tradeoff: more words, more time — so use it where thinking actually matters, not for a simple lookup.

ANALOGY

"What's 17 × 24?" — answer instantly, out loud, and most people fumble. Hand them paper to show their work and they nail it. The words the model writes are its paper.

USE IT IN YOUR BUILD

Any real judgment call — which lead to chase, how to handle a delicate client reply — tell the model to reason it through before answering, and its judgment noticeably sharpens.

> *Control the form, not just the content. Hand the model a template to fill, and it stops being a chat box and becomes a reliable software component.*

By default the model is chatty and unpredictable — three sentences today, a paragraph tomorrow, a "Sure! Here's what I found" on top. Fine for conversation, useless when a program or a brand standard needs an exact, repeatable form. The fix: tell it precisely what shape you want, and better yet, hand it the shape to fill. It works because the model just follows the groove you set — the most natural continuation is to fill your format.

Three moves make it airtight: be explicit ("exactly three bullets," "only JSON in this shape"); show the skeleton and let it fill the blanks; and forbid the noise ("no preamble, just the result"). Structured output — clean JSON your code can read — is what turns the model into a *component* you wire into a system. It's exactly how agents work: a tool call is just a precise structured output. Pair it with 4.2 — reason first, then emit the final answer in a marked block — for thoughtful results you can still parse.

GIVE IT A FORM TO FILL — AND THE OUTPUT BECOMES MACHINE-READABLE

USE IT IN YOUR BUILD

Restore Paver estimates in your exact template and voice, every time. CRM-ready lead summaries in the same fields. That consistency is a feature you can sell.

> *RAG is only as good as what it retrieves. The model will confidently use whatever you put in front of it — so retrieval quality is the whole game.*

The model is a brilliant expert who answers from whatever papers you slide over. Slide the right page, it's gold. Slide the wrong page, it gives a polished, confident, *wrong* answer and sounds just as sure. It can't tell good context from bad — so the system that fetches the pages matters as much as the genius answering.

The craft: **chunk** along natural seams (sections, Q&A pairs) — too big is noisy, too small loses meaning. **Search by meaning** with embeddings, then re-rank or blend in keyword search so the truly relevant chunk rises to the top. And the single most important instruction — the one that buys trust: tell the model to answer *only* from the retrieved material, and to say "I don't know" if it isn't there. Add **citations** so every answer is checkable. Retrieval is just context engineering on autopilot.

ANALOGY

A genius who'll answer from whatever papers you hand them. Get the librarian wrong, and the genius hands you garbage — with a perfectly straight face.

USE IT IN YOUR BUILD

For a client's customer-facing bot: clean chunking + good retrieval + "answer only from these docs, or admit you don't know." That discipline is what makes it safe to ship.

> *Building a good agent is about constraint. Manage it like a sharp but green new employee: clear goal, the right few tools, guardrails, and a human gate on anything that bites.*

An agent is a model in a loop with tools (2.3) — but every step is still prediction, so errors compound: one wrong step poisons the context for all that follow. A loose agent wanders, loops, grabs the wrong tool, or confidently takes a wrong action — automatically, maybe many times. The more autonomy, the more risk. So the whole craft is one word: constraint. Scope tightly, fail safely.

Concretely: give it a **narrow goal**, not "run my business." Give it the **right few tools** — every extra one is another way to go wrong — and describe them clearly, because the model picks tools from the descriptions *you* wrote (they're context). Put **guardrails** on it, and above all a **human gate** on anything irreversible — sending, paying, deleting, publishing. **Log** every step. And know when *not* to use an agent at all: if a single prompt or a fixed script does the job, use that.

ANALOGY

A capable new hire — fast, tireless, a little overconfident. A good manager doesn't hand over the keys to everything; they give one clear task, the right access, clear rules, and they check the important work before it ships.

USE IT IN YOUR BUILD

Your weekly human review in the content engine is exactly this instinct — the single most important guardrail. Speed everywhere else; a gate on the things that bite.

REFERENCE

Glossary

Weights	The billions of tunable numbers inside a network — where its "knowledge" actually lives.
Backpropagation	Working backward from a mistake to nudge every weight toward a better answer.
Gradient descent	Repeatedly stepping "downhill" toward less error — the core of training.
Token	A word-piece. Models read and write in tokens, and you're billed by them.
Temperature	The randomness dial at generation time — low is focused, high is creative.
Hallucination	A confident, fluent, wrong answer — the model producing a plausible next word it can't back up.
Self-attention	Every word weighing every other word to decide what's relevant. The heart of the Transformer.
Transformer	The 2017 architecture (the "T" in GPT) under nearly every modern model.
Prompt	The text you hand the model — its entire world for that turn.
RAG	Retrieval-augmented generation — fetching the right facts and adding them to the context before answering.
Embeddings	Meaning turned into coordinates, so similar ideas sit near each other.
Agent	A model in a loop with tools: think, act, observe, repeat.
Fine-tuning	Extra training on your own examples to change a model's behavior (not its knowledge).
Eval	A repeatable test that scores whether a model or prompt is actually any good.
Context window	The finite "workbench" of text the model can see in one call.

THIS MANUAL GROWS

This is , current through Part IV. As we cover each new lesson, it gets added here — new modules, new diagrams, deeper craft. A living field guide.