
TWENTY WORDS THAT
EXPLAIN MODERN AI

THE
AI
PRIMER

The twenty ideas behind every model,
explained until they click.

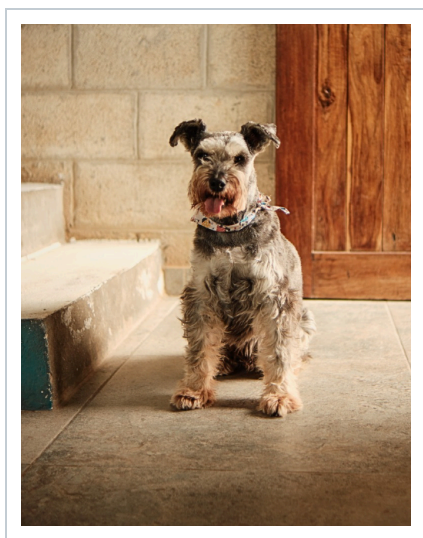
20 TERMS · PLAIN ENGLISH · NO MATH REQUIRED

PRITHVI DATLA
KILOBYTE COLLECTIVE

THE AI PRIMER



TWENTY WORDS THAT EXPLAIN MODERN AI



To a lifetime with him. The Dog in me.

The AI Primer

Published by Kilobyte Collective, 2026. Written by Prithvi Datla.

This is a plain-English field guide to the twenty words that carry modern artificial intelligence. No prior background is assumed and no mathematics is required. Every idea is built from something you already understand and then named for what it truly is.

Free to forward, quote, and cite with attribution. Corrections are welcome and will be published as amendments rather than as silent edits.

Before we begin

This book started with a phone call from my brother-in-law. He is a seasoned cybersecurity VP at a big telecom, a career spent close to hard technical problems, which is exactly why the questions caught me off guard. We were catching up about nothing in particular, and then the conversation drifted to AI, the way every conversation seems to now, and the questions started coming. How do tokens work? What is tokenization, really? How does a neural network actually do what it does?

They were good questions, and that was the whole point. If someone who lives in technology every day still wanted these answered plainly, plenty of other people did too. But the honest answers all had the same problem. Every explanation I reached for used three more words that needed explaining. You cannot define *tokenization* cleanly without leaning on *embeddings*, and you cannot get through *embeddings* without *vectors*, and pretty soon you have handed someone a tangle instead of an answer.

And he was not the first to ask. Friends ask me these things at dinner. People at work ask in the hallway after the meeting, once the room has cleared and it is finally safe to admit they were nodding along without being sure. So I decided to sit

down and write the answers out, once, in plain language, in the order the ideas actually build on one another.

I gave myself one rule that shapes the whole book. Instead of a fresh metaphor for every term, which is its own kind of tangle, I would use a single running story and let all twenty ideas live inside it. So we spend these pages inside a small coffee shop called Blackbird. A coffee shop turns out to have everything a language model has: a menu of words it knows, regulars whose patterns it has learned, a new hire you have to train, and a battered recipe binder it checks when memory runs short. Each time a real term arrives, you will find you already understood the thing. You were only ever missing its name.

Read it in order the first time. After that, keep it on the shelf and jump to whatever word your next conversation throws at you.

Prithvi Datla

KILOBYTE COLLECTIVE

The Twenty Words

01	The Large Language Model	the machine underneath
02	Tokenization	how it reads
03	Vectors & Embeddings	meaning as numbers
04	Attention	what it looks at
05	Self-Supervised Learning	teaching itself
06	The Transformer	the shape of the thing
07	Fine-Tuning	giving it a job
08	Few-Shot Prompting	teaching by example
09	Retrieval-Augmented Generation	looking things up
10	The Vector Database	where meaning is stored
11	Model Context Protocol	a common plug
12	Context Engineering	what you put in front of it
13	Agents	when it can act
14	Reinforcement Learning	learning from feedback
15	Chain of Thought	thinking out loud
16	Reasoning Models	thinking before it answers
17	Multimodal Models	beyond text
18	Small Language Models	less, on purpose
19	Distillation	a student and a teacher
20	Quantization	making it fit

The Large Language Model

Nadia has read almost everything ever written. Ask her anything and a fluent answer comes back, in whole sentences. She is not looking it up.

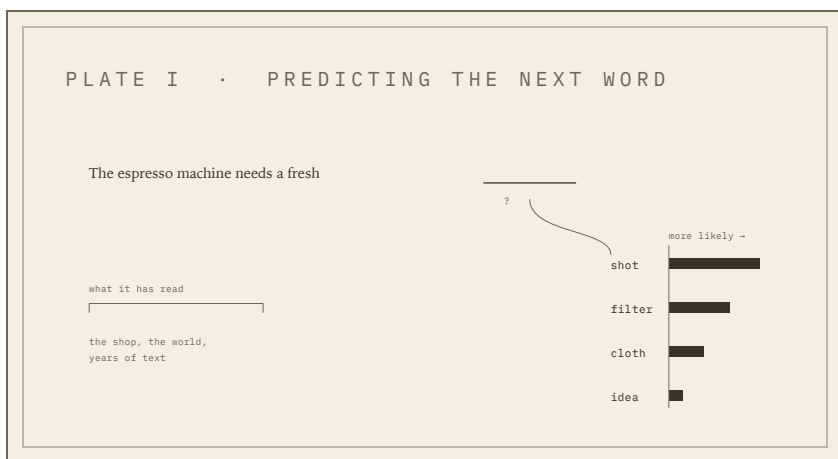
Blackbird Coffee has a new hire named Nadia, and Nadia has an unusual history. Before her first shift she somehow read almost everything anyone has ever written down. Every cookbook and coffee manual. Every forum argument about grind size. Novels, menus from ten thousand other shops, years of newspapers, the whole sprawl of it. Blackbird's own binder is in there too, but only as a thin folder in an ocean.

So when a regular walks up and says *the usual, but make it warmer today*, Nadia does not freeze. She has seen a thousand sentences that start like that, and she has a feel for how they tend to end, so a sensible answer comes out of her almost on its own. Not because someone handed her a rule for that exact request, but because she has read so much that the next word simply arrives. She does not yet know Blackbird's particular quirks, the

way this shop and no other likes things said. That comes later. For now she is fluent, and fluent is already a great deal.

That feel, built at an enormous scale, is a large language model.

Strip away the mystery and the machine is doing something almost dull. It was trained on a staggering amount of text with a single repetitive task: given the words so far, guess the next chunk of text. Guess, check against what actually came next, adjust, and repeat. Do that a few hundred billion times and the millions of little internal settings inside the model, its *parameters*, slowly arrange themselves into a compressed map of how language tends to go, which is to say a map of how we describe the world.



The model never chooses a whole answer at once. It weighs which word most plausibly comes next, commits to one, then does it all again for the word after that.

Large is meant literally. It points at two numbers, both enormous: how much text the model read while training, and how many of those internal settings it carries, now routinely in the billions. Make both bigger and the feel gets sharper. That is most of what people mean when they say a model is more capable.

Here is the part that trips everyone up. There is no filing cabinet inside the model, and it is not a search engine. When Nadia answers, she is not flipping to the page where that exact sentence is written. She is producing it, word by word, each word leaning on every word that came before. The model works the same way. Its fluency is not retrieval. It is the side effect of getting very, very good at one prediction game.

Everything else in this book is a trick wrapped around this one machine. How it reads its input, how it holds meaning, what it pays attention to, how we give it a job, how we let it look things up or take actions. Twenty words, but they all circle back to the next-word predictor you just met behind the Blackbird counter.

IN ONE SENTENCE

A large language model is a next-word predictor trained on so much text that predicting well forces it to absorb how the world is described.

Tokenization

Nadia does not read your order letter by letter, or even word by word. She reads it in familiar chunks she has seen a thousand times.

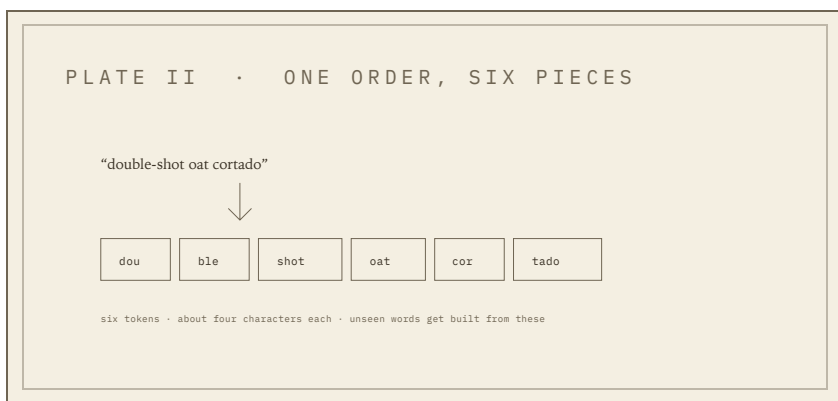
Watch closely as a ticket comes in reading *double-shot oat cortado*. Nadia does not take it in as a run of letters, and not quite as three whole words either. She sees a small handful of familiar pieces: *dou*, *ble*, *shot*, *oat*, *cor*, *tado*. Pieces she has met countless times before. Those pieces have a name. They are tokens.

Why chop things up at all? Because a fixed, manageable set of pieces lets her handle anything, including words she has never once seen. A customer invents *cortadoccino* on the spot and Nadia does not stall. She builds it from scraps she already knows: *cor*, *tado*, *ccino*. Common words tend to be a single token. Rare or made-up ones get assembled from smaller stubs. Nothing is ever truly foreign; it is only spelled out of familiar parts.

So a token is not a letter and not always a whole word. It sits in between, a common chunk of text. A loose rule for English: a token runs about four characters, which means a hundred

tokens is roughly seventy-five words. Spaces and punctuation get their own pieces too. The exact set of tokens is decided once, up front, by scanning a mountain of text and keeping whichever chunks show up most.

This matters more than it first appears, because everything a model does is counted in tokens. When you hear that a model has a context window of two hundred thousand tokens, that is how many of these pieces it can hold in view at once. When a service charges by the token, this is the unit ticking on the meter. And here is the honest correction to chapter one: the model never really sees your sentence. It sees the sequence of tokens your sentence was chopped into, and it predicts the next token, not the next word. The next-word predictor is, underneath, a next-token predictor.



The model reads and bills in these pieces, not in words. A word it has never seen is simply assembled from smaller ones it has.

IN ONE SENTENCE

Tokenization chops text into a fixed menu of common pieces, so a model can read anything and measure everything in the same small units.

Vectors & Embeddings

Nadia keeps everything she knows on one enormous pegboard, where things that mean similar things hang close together.

A pile of tokens is not yet understanding. To make sense of them, Nadia needs them organized, and she does not file them alphabetically. Alphabetical order would hang *cat* beside *cattle* beside *catastrophe*, words that share some letters and no meaning at all. Instead she hangs every word on a vast pegboard arranged by meaning. *Latte* hangs a hand's width from *cappuccino*. *Oat milk* sits in the little cluster of milks. *Wrench* hangs clear across the room.

A position on that pegboard is just a list of coordinates. On a real pegboard you would need two numbers, how far across and how far up. Nadia's board runs in hundreds of directions at once, so each word's position is a list of hundreds of numbers. That list is a *vector*, and the learned position of a word is its *embedding*. When you hear that a model turns words into vectors, this is the pegboard it is placing them on.

The quiet magic is not that words get numbers. It is that the geometry carries meaning, so that directions on the board come to mean things. The step from *espresso* to *double espresso* is roughly the same step as from *coffee* to *large coffee*: a “more of it” direction you can actually travel. Word arithmetic falls out of this. Start at the point for *hot latte*, take the short “make it iced” step that separates every warm drink from its cold twin, and you land almost exactly on *iced latte*. Meaning is stored as place, so moving through the space is moving through meaning.

This is why a model can tell that “the café was slammed” and “the shop got swamped” are saying one thing though they share almost no letters. On the pegboard they land in nearly the same spot. Embeddings are a model’s sense that different words can point at a single idea, and one word can point at many. Every token from the last chapter gets turned into its vector before any real thinking begins. Meaning, for a machine, is a location.

PLATE III · THE PEGBOARD OF MEANING



Each word is a point, its position a list of numbers. Nearness on the board is nearness in meaning, which is why the coffees huddle and the wrench sits alone.

IN ONE SENTENCE

An embedding is a word's position in a space of meaning, written as a list of numbers, so that closeness in the space means closeness in meaning.

Attention

“Make the second one decaf.” Which one? Nadia has to look back at the order to know, and knowing what to look back at is the whole trick.

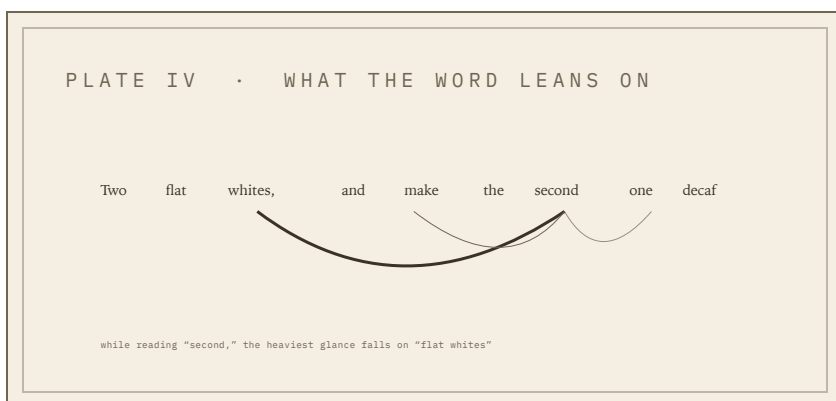
A regular orders: “Two flat whites, and make the second one decaf.” To pour it right, Nadia has to connect *second one* back to *flat whites*, four words upstream, and not to *decaf* or *and*. Get that thread wrong and someone’s evening is ruined. The meaning of a word depends on which other words it is really about.

For most of computing history, machines read strictly left to right and were poor at holding those threads across a sentence. The breakthrough, the one that set off nearly everything you now call AI, was a mechanism that lets the model, for each word it is working on, glance back over all the other words and decide how much each one matters right now. That weighing is *attention*.

Picture Nadia as she considers *second*. She casts a glance across the whole sentence and lets it rest heaviest on *flat whites*, lightly on *two*, barely at all on *and*. A different word draws a different

glance: when she reaches *decaf* she looks hardest at *second one*. Attention is that glance, computed for every word against every other word, and turned into a set of weights that say what to lean on.

This solved the thing that made language so hard for machines: meaning leans on distant context, and which distant context matters changes word by word. *It* in “the mug was hot so I put it down” points at the mug; shift a few words and it points elsewhere. Attention is how a model works out, freshly each time, what each word is truly about. The 2017 paper that introduced this at scale was titled, with some confidence, “Attention Is All You Need.” It was close to right.



For each word, the model weighs every other word and leans on the ones that matter. The thick arc is a strong link; the faint ones barely register.

IN ONE SENTENCE

Attention is how a model decides, for each word, which other words to lean on, so meaning built from distant context comes out right.

Self-Supervised Learning

Nadia trained herself, with no teacher and no answer key. She covered the last word of real tickets, guessed it, then peeked.

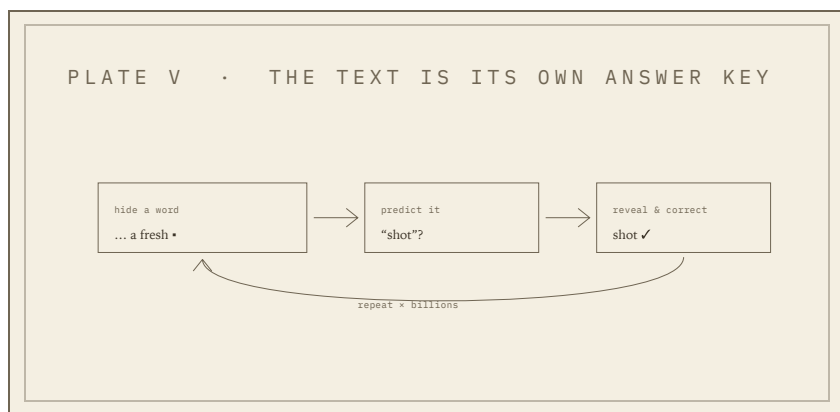
Here is a fair objection to chapter one. To learn from all that text, surely someone had to sit down and mark it, label the right answers, grade the work. For a single shop's tickets, maybe. For the whole internet, never. There are not enough people alive to hand-grade that much writing.

The trick that made large models possible is that the text grades itself. Take any real sentence the shop ever wrote. Cover the last word. Ask Nadia to guess it. Then uncover it and see whether she was right. The correct answer was sitting there in the text the entire time; you only hid it and revealed it. No human wrote an answer key, because every real sentence is already its own answer key.

Do that across billions of sentences, hiding a word here and a chunk there, nudging the parameters a hair closer each time the guess misses, and you grow chapter one's uncanny feel for

language without anyone labelling a thing. That is self-supervised learning: the supervision, the supply of right answers, comes free from the structure of the data itself.

The distinction is worth keeping. Old-style supervised learning needs humans to label examples, *this is a cat*, *this is not*, and that human effort is the ceiling on how much a model can learn. Self-supervised learning removes the ceiling. The reason models turned shockingly good once training was scaled up around 2020 is partly clever design and mostly this: we found a way to learn from essentially all the text there is, without paying anyone to grade it.



No one writes the answers down. You hide part of a real sentence, let the model guess, then reveal what was always there, and repeat until fluency emerges.

IN ONE SENTENCE

Self-supervised learning lets a model teach itself from raw text by hiding parts and predicting them, so the right answers come free and the examples never run out.

The Transformer

Attention was the idea. The transformer is the machine built around it, and it is the shape of nearly everything you now call AI.

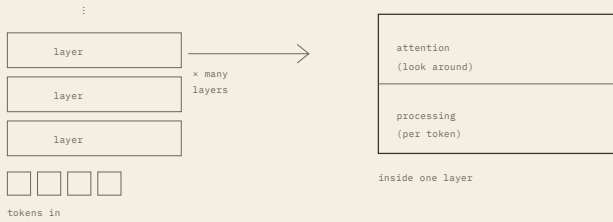
By now Nadia has a few distinct talents. She reads in tokens, she places every token on the pegboard of meaning, and she can pay attention across a sentence. The transformer is the actual machine that arranges those talents into something that works at scale. It is the blueprint. The *T* in GPT, and in most model names you will meet, is this one word.

Think of it as an assembly line with many identical stations. Your tokens enter as embeddings. At each station two things happen: every token looks around and attends to the others, then each token passes through a small bit of private processing that updates what it represents. Then it moves to the next station, which does the same again. Stack dozens of these, sometimes well over a hundred, and by the end each token's meaning has been enriched by everything around it, refined layer upon layer.

What makes the transformer special is not any single station. It is that all the tokens travel through together, in parallel, rather than one after another. Earlier designs shuffled words through in strict sequence, which was slow and forgetful across long distances. The transformer takes in the whole passage at once. That parallelism is precisely why these models could be trained on oceans of text using modern hardware; the older shapes simply could not scale.

It arrived in that 2017 paper and quietly became the ground the whole field stands on. Almost every model in this book, the chat assistants, the image generators, the coding tools, is a transformer or a close relative. When people speak of “the architecture” behind modern AI, this is the thing they mean. Everything before this chapter was a talent. The transformer is the body those talents finally live in.

PLATE VI • A STACK OF IDENTICAL LAYERS



Tokens flow up through a tall stack of identical layers. Each layer lets every token attend to the others, then refines it, over and over.

IN ONE SENTENCE

The transformer is the assembly-line architecture that runs attention across all tokens in parallel, stacked in many layers, and it underlies almost every modern model.

Fine-Tuning

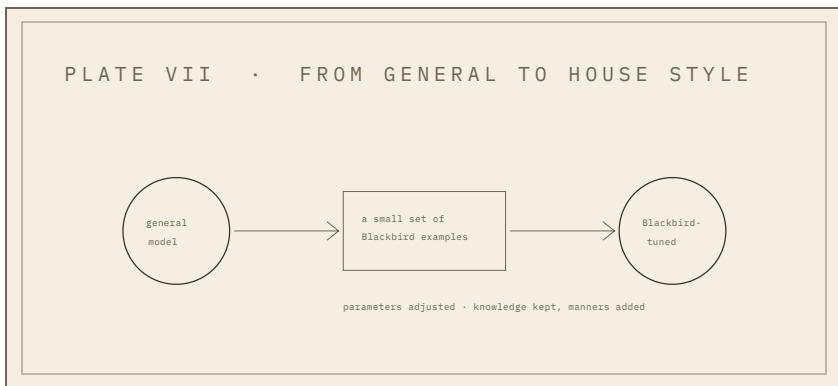
A model that has read everything knows a little about all of it and the house style of nothing. Fine-tuning is the week you spend teaching it your shop.

Fresh out of training, Nadia is broadly capable and generically mannered. She has read enough to discuss almost anything, but she does not yet sound like Blackbird. She does not know that here we never say “no problem,” we say “of course”; that refunds get a warm, short apology and one specific fix; that the owner cannot abide an exclamation mark. She has the general ability. She lacks the job.

Fine-tuning is the focused stretch of training that gives her the job. You take the finished, general model and keep training it, now on a smaller, carefully chosen pile of examples that show the behavior you want: hundreds or thousands of Blackbird-style exchanges. The broad knowledge stays; the manners and the specialty settle in on top. It is the difference between hiring someone who can obviously do the work and spending a week showing them exactly how it is done here.

The reason anyone bothers, rather than training a model from scratch, is cost. Building the general ability took an ocean of text and a fortune in computing. That part is done, and it is reusable. Fine-tuning reshapes an existing model with a comparatively tiny amount of data and effort. It is how one giant base model becomes a thousand specialists: a medical one, a legal one, one that answers only ever in your company's voice.

Keep the boundary sharp, because the next chapter presses right up against it. Fine-tuning changes the model itself. You are editing its parameters, leaving the shop with a hire who has genuinely absorbed the house style and will carry it into tomorrow without being reminded. That permanence is the whole point, and it is exactly what the next idea does without.



The finished model keeps training on a focused pile of examples. Its broad knowledge stays; a particular voice and job settle in on top.

IN ONE SENTENCE

Fine-tuning continues training a finished model on a small, focused set of examples, permanently adjusting it to a particular job or voice.

Few-Shot Prompting

Sometimes you do not retrain Nadia at all. You show her two examples, then ask. She takes the pattern and runs with it.

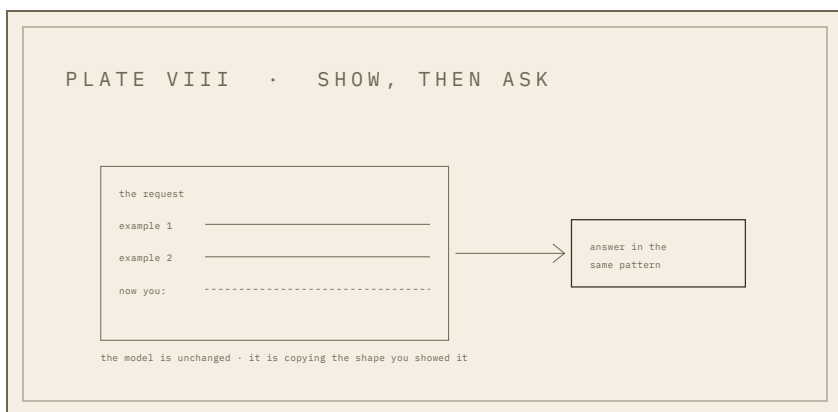
Fine-tuning was a week of teaching. Few-shot prompting is a moment. Suppose you do not want to retrain Nadia, you just need her to word a refund note the way you like, right now. You could describe the style at length, or you could simply show her: “Here is one we sent last week. Here is another. Now write one for table four.” Two examples, and she has the pattern.

That is few-shot prompting: giving the model a handful of worked examples inside your request, so it can infer the pattern and match it, with no training at all. Nothing about the model changes. You are steering a fixed model in the moment by loading its input with the shape of the thing you want. Show it the form of a good answer and it will tend to produce that form.

The names simply count the examples. *Zero-shot* is asking cold, with none, “write a refund note,” and trusting the model’s general ability. *One-shot* gives a single example. *Few-shot* gives

several. More examples usually mean a closer match, up to a point, and they cost you room in the request, which the earlier chapters told you is measured in tokens and is finite.

Hold onto the contrast: fine-tuning versus few-shot is training versus showing. Fine-tuning edits the hire and lasts. Few-shot leaves the hire untouched and lasts exactly as long as the conversation. Most of the clever prompting you have heard about is, underneath, someone discovering that one good example in the request beats a paragraph of instructions.



A couple of worked examples placed inside the request are enough for the model to infer the pattern and match it, without any training.

IN ONE SENTENCE

Few-shot prompting steers a fixed model by placing a few worked examples in the request itself, so it copies the pattern without any retraining.

Retrieval-Augmented Generation

Nadia's memory is vast but frozen at her first day, and she will not admit when she is unsure. The recipe binder fixes both.

For all her fluency, Nadia has two blind spots. Her knowledge froze the day her training ended, so she has never heard of this morning's special or last week's price change. And when she does not know, she does not always go quiet; she can produce a confident, well-worded answer that happens to be wrong. In AI this is politely called *hallucination*, and it is the same machinery from chapter one doing exactly its job: predicting a plausible next word, whether or not it is true.

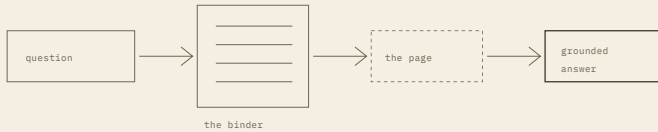
The fix is refreshingly ordinary. Before Nadia answers a question of fact, she checks the binder. A customer asks, "what is in the autumn latte this year?" and instead of reciting from frozen memory, she flips to today's recipe page, reads it, and answers from what is in front of her. That is retrieval-augmented generation: retrieve the relevant text first, hand it to

the model along with the question, and let it generate an answer grounded in that text.

The word *augmented* is the point. You are not retraining the model and you are not replacing it. You are augmenting its fixed general ability with fresh, specific, trustworthy pages pulled in at the last second. The model still writes the answer in its own fluent voice; it just writes it about the binder page rather than about whatever it half-remembers. Update the binder and the answers update at once, no retraining required.

This is how most of the useful AI systems you will meet actually work. The chatbot that answers questions about your company's policies is not a model that memorized the handbook; it is a general model that, on each question, is quietly handed the relevant handbook pages and asked to answer from them. It is the difference between "tell me from memory" and "here, read this and then tell me," and the second is far more reliable.

PLATE IX · LOOK IT UP, THEN ANSWER



The question first pulls the relevant page from the binder. Only then does the model answer, writing about the page in front of it rather than from frozen memory.

IN ONE SENTENCE

Retrieval-augmented generation looks up relevant text first and hands it to the model with the question, so answers stay current and grounded instead of guessed from memory.

The Vector Database

The binder only helps if Nadia can find the right page in a heartbeat, and she cannot do it alphabetically. She finds it by meaning.

Retrieval has a hidden hard part. “Check the binder” is easy to say, but the binder holds thousands of pages, and the customer’s question almost never uses the same words as the right page. Someone asks “is the autumn drink dairy-free?” and the page that answers is titled “Maple Oat Latte, seasonal, made with oat milk.” No shared keywords. Search the old way, by matching words, and Nadia comes up empty while holding the exact answer.

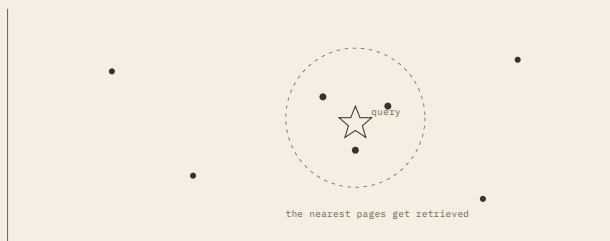
The fix reaches back to chapter three. Remember the pegboard, where things that mean similar things sit close together. Store every binder page by its position on that pegboard instead of by its words. Now the question “is the autumn drink dairy-free?” lands, as a position, right beside the oat-milk page, because they mean nearly the same thing even though they share no

letters. To find the right page you no longer match words; you go to where the question lands and grab whatever is nearby.

A vector database is the filing system built for exactly that. It stores millions of these positions, the embeddings from chapter three, and it is very good at one question: given a new position, what is closest to it? That “find me the nearest neighbors” search, at huge scale and in milliseconds, is the whole reason the thing exists. It is the binder’s index, organized by meaning rather than by alphabet.

So the two chapters click together. Retrieval is the strategy: look it up before you answer. The vector database is the machine that makes the lookup work, by searching on meaning instead of on matching words. Nearly every “chat with your documents” product is these two ideas holding hands: embed everything into a vector database, and on each question, retrieve the nearest pages and hand them to the model.

PLATE X · NEAREST BY MEANING



The question becomes a point. The database returns whatever sits nearest to it, matching on meaning even when the words do not line up at all.

IN ONE SENTENCE

A vector database stores text by its position in meaning-space and finds the closest matches to a query, which is what makes retrieval work when the words do not line up.

Model Context Protocol

Nadia can read the binder, but the register, the supplier's system, and the calendar each speak their own language. This is one plug that fits them all.

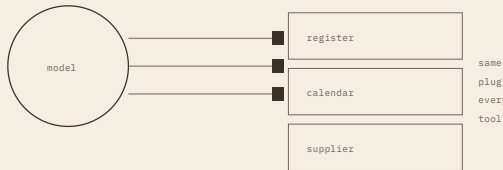
Nadia grows far more useful the moment she can reach the shop's other systems: the register to check a price, the supplier's site to reorder beans, the calendar to see who is on shift. The trouble is that each was built by different people and speaks its own dialect. Wiring Nadia to each one separately is a custom job every time, and it breaks whenever any system changes. Multiply that across every AI assistant and every tool in the world and you get a spaghetti of one-off connectors.

The Model Context Protocol is a standard plug. It is an agreement on how a model and an outside tool talk to each other, so that any model speaking the protocol can connect to any tool that also speaks it, with no custom wiring. Think of what USB did for devices. Before it, every gadget had its own cable; after it, one shape fit them all. This aims to be that shape for connecting AI models to tools, data sources, and systems.

Concretely, a tool exposes itself through the protocol by describing what it can do, “I can look up a price, given an item,” “I can place an order, given a quantity,” in a form the model understands. The model, needing a price, sees that capability advertised and uses it through the same standard channel it uses for every other tool. The register, the supplier, and the calendar all become things Nadia can reach the same way, instead of three separate wiring projects.

It is an unglamorous idea and a load-bearing one. As models move from answering questions to actually doing things, the next chapter, the bottleneck stops being the model and becomes the plumbing between the model and everything it needs to touch. The Model Context Protocol is an attempt to standardize that plumbing so it stops being rebuilt from scratch every single time.

PLATE XI · ONE PLUG FOR EVERY TOOL



Instead of a custom cable per system, one agreed connector lets any model reach any tool that speaks the same protocol.

IN ONE SENTENCE

The Model Context Protocol is a standard connector that lets any model talk to any tool or data source through one common interface, instead of custom wiring for each.

Context Engineering

Give Nadia a bare question and she guesses. Set the right ticket, the right binder page, and a note about the customer in front of her, and she is brilliant.

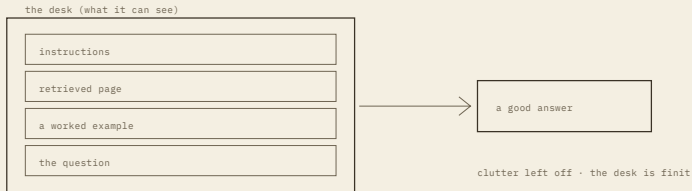
Everything a model knows in the moment of answering, it knows because it is sitting right in front of it. Chapter two named the unit, tokens; the whole span of tokens a model can see at once is its *context*, its working desk. Whatever is on the desk shapes the answer; whatever is not may as well not exist. Context engineering is the craft of deciding what goes on that desk.

Watch it happen at Blackbird. A customer asks Nadia to “sort out my usual mess from last time.” Handed just that, she is lost. But set the right things on her desk first, the customer’s order history retrieved from chapter nine, the note that “mess” meant a spilled oat latte, a short instruction that we comp the redo and apologize once, and the same model that would have floundered now handles it perfectly. Nothing about Nadia changed. Her desk did.

This has quietly become where much of the real work lives. Getting good answers from a model is less about clever wording and more about assembling the right context: the relevant retrieved pages, a few worked examples from chapter eight, the necessary instructions, the history that matters, and, just as important, leaving off the clutter that would distract or crowd the desk out. The desk is finite, remember, it is measured in tokens, so context engineering is also the art of what to leave off.

It is worth separating from its noisier cousin. Prompt engineering is fussing over the wording of the question. Context engineering is the larger discipline of arranging everything around the question: what to retrieve, what to show, what to remember, what to omit. As systems grow more capable, this is increasingly the job, not coaxing a reluctant model, but setting its desk so well that the right answer is nearly inevitable.

PLATE XII · SETTING THE DESK



The same model, given a well-set desk, answers what it could not before. The craft is choosing what to place on it and what to keep off.

IN ONE SENTENCE

Context engineering is the craft of assembling exactly the right information on a model's working desk, so the answer follows almost inevitably from what it can see.

Agents

Until now Nadia only answered. An agent Nadia decides she needs beans, walks to the storeroom, places the order, checks it arrived, and only then reports back.

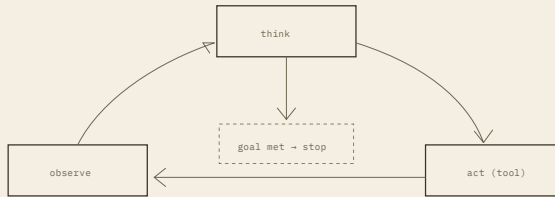
Every Nadia so far has been a talker. You ask, she answers, and the loop ends there. An *agent* is what you get when you let her act, and then loop. Give her a goal instead of a question, “we are low on oat milk, handle it,” and she does something, sees what happened, decides the next thing, and keeps going until the goal is met. Answering becomes doing.

The machinery is the model from chapter one plus three new privileges: tools, a goal, and a loop. The tools are the reach from chapter eleven, the register, the supplier, the calendar. The goal is the standing instruction. The loop is the crucial part: the model looks at the situation, chooses a tool and uses it, reads the result, and feeds that result back to itself to decide the next move. Order the oat milk, see the confirmation, notice the delivery is Thursday, leave a note for Thursday’s shift. Decide, act, observe, repeat.

That loop is exactly what separates an agent from everything earlier in the book. A plain model is a single step: text in, text out. An agent runs many steps of its own accord, each one informed by what the last one turned up, choosing along the way which tools to use and when it is finished. It can chain together actions you never spelled out, because at each turn it is deciding the next move rather than following a fixed script.

This is also where the field's ambition and its anxiety both live. An agent that takes real actions in real systems is enormously useful and genuinely riskier than a chatbot; a wrong answer is a sentence, a wrong action is an order actually placed. Most of what you now hear called "AI agents" is this pattern, a model in a loop with tools and a goal, and most of the engineering around it is about keeping that loop capable without letting it run off the rails.

PLATE XIII · A MODEL IN A LOOP



The agent thinks, uses a tool, observes the result, and thinks again, looping under its own steam until the goal is reached.

IN ONE SENTENCE

An agent is a model placed in a loop with tools and a goal, so it can decide, act, observe the result, and act again until the job is done.

Reinforcement Learning

Nadia learned language by prediction, but not manners. Manners she learned from the tip jar: do the thing that earns thanks, avoid the thing that earns a frown.

Prediction, back in chapter five, made Nadia fluent. It did not make her helpful, or honest, or pleasant to deal with. A pure next-word predictor trained on the whole internet will happily continue a rude sentence rudely, because rude sentences exist and it has only ever been asked to guess what comes next. Something has to teach it not what is *likely*, but what is *wanted*. That something is reward.

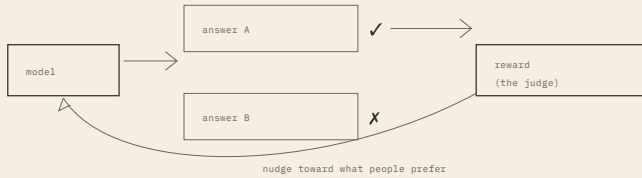
Reinforcement learning is training by consequence rather than by example. You do not hand the model the right answer; you let it act, then give it a score, higher for outcomes you want and lower for ones you do not, and it gradually shifts toward whatever earns the high scores. It is the tip jar, not the answer key. Nadia tries a way of handling a complaint, sees the tip and the smile, and drifts toward doing more of that. No one wrote

down the perfect complaint response; she found it by chasing reward.

The version that shaped the assistants you actually use has a longer name: reinforcement learning from human feedback, or RLHF. The problem it solves is that “a good answer” is too fuzzy to score with a formula. So people are shown pairs of the model’s answers and simply pick the better one, many thousands of times. Those preferences train a second model, a stand-in judge that has learned the collective taste, and that judge then hands out rewards at scale. The main model practices against the judge, tuning itself toward answers people prefer.

This is the step that turned raw, fluent, occasionally feral predictors into the polite, helpful assistants that broke into the mainstream in late 2022. The underlying knowledge came from prediction; the helpfulness, the willingness to refuse a bad request, the house manners, came from reward. Nadia learned to talk by reading. She learned how to behave from the tip jar.

PLATE XIV · LEARNING FROM THE TIP JAR



People pick the better of two answers; those choices train a judge; the judge's rewards steer the model toward answers people want.

IN ONE SENTENCE

Reinforcement learning trains a model by rewarding good outcomes rather than showing right answers, and RLHF uses human preference as that reward to make a model helpful.

Chain of Thought

Ask Nadia a tricky total in her head and she blurts a wrong number. Ask her to work it out on the ticket, step by step, and she gets it right.

Here is a strange, true thing about the next-word predictor. Ask it a hard, multi-step question and demand only the final answer, and it often fumbles, because it is trying to leap straight to the last word with no room to work. Ask the very same model to “think step by step” first, and its accuracy on hard problems jumps, sometimes dramatically. Same model, same knowledge; the only change is that you let it show its work.

Picture a fiddly order: three drinks, one with an upcharge, a member discount, split two ways. Ask Nadia for the per-person total cold and she will guess. Ask her to write it out, drinks come to this, add the upcharge, take the discount, divide by two, and each written step gives the next one something solid to stand on. Chain of thought is exactly that: prompting the model to generate its reasoning in steps before its answer, instead of vaulting straight to the end.

Why it works follows straight from chapter one. The model produces one token at a time, each leaning on the tokens before it. When it writes out its reasoning, those intermediate steps become part of what it is reading as it goes, so each step props up the next. Skip the steps and it has nothing to lean on but the bare question. The written thinking is not decoration; it is scratch paper the model then gets to read.

The practical upshot is almost too simple: for anything with steps, arithmetic, logic, planning, multi-part instructions, letting the model reason out loud beats forcing a snap answer. It costs more tokens and a moment more time, and it buys a real jump in getting hard things right. This one small idea, give the model room to think on the page, is the seed of the whole next chapter.

PLATE XV · WORKING IT OUT ON THE PAGE

straight to the answer

= \$9.40 ✗

step by step

1 · drinks → 12.00
2 · upcharge → 13.00
3 · discount → 11.70
4 · split two → 5.85

= \$5.85 ✓

Forced to answer at once, the model guesses. Allowed to lay down steps, each line supports the next and the final figure comes out right.

IN ONE SENTENCE

Chain of thought is prompting a model to write its reasoning step by step before answering, which gives each step something to build on and sharply improves hard problems.

Reasoning Models

Chain of thought was a trick you asked for. A reasoning model is a Nadia trained to always think it through, privately and at length, before she says a word.

Chain of thought, in the last chapter, was something you had to request, and the thinking spilled out into the answer. Reasoning models take that lesson and build it into the model itself. These are models trained, not merely prompted, to think, to spend real effort deliberating before they answer, especially on hard problems. You do not have to say “think step by step”; deliberation is the default, whether or not you ask.

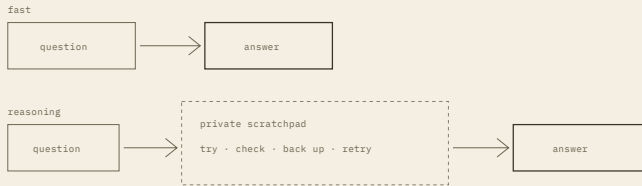
The analogy sharpens if you imagine two hires. One answers the instant you finish asking, fast and usually fine. The other pauses, works the problem through on a private notepad you never see, weighs a couple of approaches, checks herself, and only then answers. For “what is a medium latte?” the pause is wasted. For a tangled, multi-step problem, the second hire is worth the wait by a mile. A reasoning model is that second hire,

and it has learned, through training, when and how to do the pausing well.

Two things set them apart from an ordinary model told to think. First, the deliberation is often hidden, an internal scratchpad the model reasons on before composing its final reply, so you see a considered answer rather than the whole mess of working. Second, and this is the deeper shift, they were specifically trained, using the reward methods from chapter fourteen, to reason well: to try an approach, notice when it is going wrong, back up, and try another. The thinking got better because it was practiced, not merely permitted.

The trade-off is blunt and worth naming. Reasoning models are slower and cost more, because thinking burns tokens and time, and for simple requests that is pure waste. But on genuinely hard work, careful math, thorny logic, real planning, they are markedly more capable than a model that answers on reflex. The field's recent leap on difficult problems is largely this: not just bigger models, but models that have learned to stop and think before they speak.

PLATE XVI · TWO WAYS TO ANSWER



An ordinary model answers on reflex. A reasoning model deliberates on a hidden notepad first, spending time where the problem is genuinely hard.

IN ONE SENTENCE

A reasoning model is one trained to deliberate at length, usually on a hidden scratchpad, before answering, trading speed for a real gain on hard problems.

Multimodal Models

The same Nadia who reads tickets can now look at the photo you hold up and listen to the order you speak. One mind, more senses.

Everything so far has been about text. But the pegboard idea from chapter three does not care what it is holding. If you can turn a thing into a position in meaning-space, the same machinery works on it, and it turns out you can do that for pictures and sound as readily as for words. A multimodal model is one that handles more than one kind of input, text and images, often audio and video too, inside a single system.

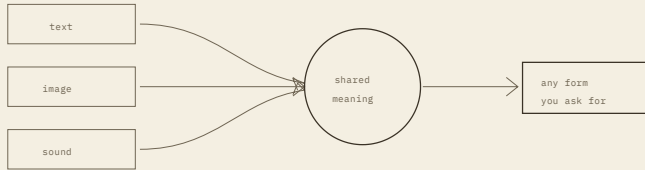
At Blackbird this is the difference between a Nadia who only reads written tickets and one who can also glance at the photo a customer holds up, “can you make this?”, and hear a spoken order over the noise. The powerful part is not that she has separate skills bolted together. It is that they share one understanding. A picture of a latte, the word *latte*, and someone saying “latte” all land in nearly the same spot on the pegboard,

so she can move between them: describe a photo in words, or find the drink that matches a picture.

That shared space is what makes multimodal models more than the sum of their parts. Because a photo of a dog and the words “a dog” map to the same neighborhood of meaning, a model can caption images, answer questions about a chart, read a handwritten note, or generate a picture from a description, all as the same act of translating between a shared representation and whatever form you asked for. The senses differ; the meaning underneath is one common currency.

This is why the assistants you use kept sprouting eyes and ears. You can show a model a photo of the inside of your fridge and ask what to cook, hand it a screenshot of an error and ask what is wrong, or speak instead of type. Under the hood it is the same transformer from chapter six, fed tokens that now stand for patches of an image or slices of sound as readily as for words. More senses, same mind.

PLATE XVII · MANY SENSES, ONE MEANING



Words, pictures, and sound are mapped into one shared space of meaning, so the model can move between them and answer in whatever form you want.

IN ONE SENTENCE

A multimodal model handles text, images, and often sound in one system by mapping them all into a shared space of meaning, so it can move freely between them.

Small Language Models

Not every job needs the Nadia who read the whole internet. For the drive-through you want a fast, cheap clerk who knows only the menu, and runs on the shop's own tablet.

The whole book has quietly assumed that bigger is better: more text, more parameters, more capability. For raw breadth, that is true. But breadth has a bill. The largest models are enormous, expensive to run, and live in distant data centers you reach over the internet, paying by the token each time. For plenty of real jobs, that is using a research library to answer “what time do you close?”

A small language model is the deliberate opposite: the same transformer machinery, but with far fewer parameters, trained or tuned to do a narrower range of things well rather than everything passably. Think of a pocket-sized Nadia who never read the whole internet, only every Blackbird menu and ticket. Ask her the capital of a country and she shrugs. Ask her to take a coffee order and she is instant, tireless, and every bit as good

as her giant cousin, because that is the only thing she needs to be good at.

Small buys you things large cannot. A small model can run on the shop's own tablet, or your phone, or a car, no data center, no internet round-trip, no per-token bill, and nothing about the conversation ever leaves the device, which matters when the input is private. It answers faster because it is near and lean. And it is cheap enough to run constantly, on every ticket, without a second thought about cost.

So the frontier is not only a race for the biggest brain. A great deal of the field's practical energy has swung the other way, toward models small enough to live inside products, on devices, at the edge, doing one family of tasks superbly. Often the right design is a team: a small model handling the constant, narrow work on-device, calling on a large one only for the rare question that genuinely needs the whole library. Right-sized beats biggest more often than the headlines suggest.

PLATE XVIII · RIGHT-SIZED, NOT BIGGEST



large model
distant data center
broad · costly · remote

small model



on your
device:
fast, cheap,
private

The same architecture, shrunk. A narrow model can live on the device itself, answering instantly and privately for the one job it was built to do.

IN ONE SENTENCE

A small language model uses the same architecture with far fewer parameters to do a narrow job well, cheaply, and often right on your device instead of a distant data center.

Distillation

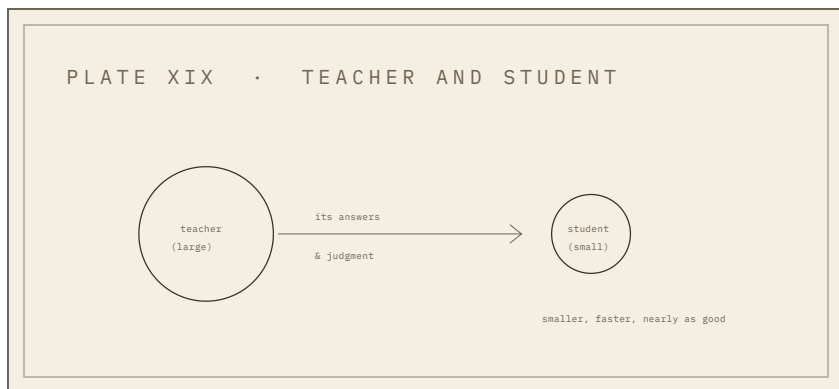
How do you get a small Nadia who is nearly as sharp as the big one? You have the veteran teach the junior, not from the manual, but by copying her answers.

The last chapter wanted small models that punch above their size. Distillation is the main way you get them. The blunt approach, training a small model from scratch on raw text, gives you something small but dim; there simply is not enough capacity to learn everything the hard way. The clever approach is to let a big, already-brilliant model do the teaching.

Picture the veteran Nadia, the one who read everything, training a junior. She does not hand over the whole library. Instead, for order after order, she shows the junior how she would answer, and the junior learns to imitate her responses. Crucially, the junior is not just copying the final answer; she is picking up the veteran's judgment, *this drink is very likely a latte, but there is a slim chance it is a cortado*, which carries far more than a bare right-or-wrong. Learning from a wise teacher's shaded answers, the junior gets most of the way to the veteran's skill at a fraction of the size.

That is knowledge distillation: train a small “student” model on the outputs of a large “teacher” model, so the student absorbs the teacher’s behavior rather than learning from raw data alone. The student ends up dramatically smaller and faster while keeping a surprising share of the teacher’s ability, because it learned from something that had already done the hard work of making sense of the world. It is the difference between learning surgery from a textbook and learning it at the elbow of a master.

This is the quiet mechanism behind a lot of the impressive small models you will meet. That fast, cheap model running in an app did not get sharp by reading less; it got sharp by studying under a giant. Distillation is how the expensive brilliance trained once into a huge model gets pressed down into something light enough to hand to everyone.



The student trains on the teacher’s shaded answers, not raw text, and inherits most of its skill in a far smaller frame.

IN ONE SENTENCE

Distillation trains a small student model to imitate a large teacher model's answers, transferring most of its ability into a fraction of the size.

Quantization

The recipe binder is written to absurd precision: 37.412 ml of milk. Round every number to something sensible and it gets far lighter while the coffee tastes the same.

One last way to shrink a model, and it is the most purely mechanical. Remember that a model is, underneath, millions or billions of numbers, its parameters, the settings tuned during training. Those numbers are normally stored at high precision, many digits each, which is accurate and heavy. Quantization is the act of rounding them off, storing each number in a coarser, smaller form, to make the whole model lighter.

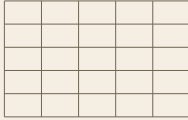
Blackbird's binder makes it plain. Suppose every recipe were written to a ridiculous exactness: 37.412 ml of milk, 18.06 grams of beans. It would be precise and enormous. Now round everything to what a person actually uses, a splash, two scoops, and the binder shrinks to a fraction of its bulk while the coffee comes out indistinguishable. The recipes are rougher on paper; no one can taste the difference. That is the bet quantization makes with a model's numbers.

In practice this means taking each parameter, often stored in sixteen bits, and squeezing it down to eight, which halves the model's size, or all the way to four, which quarters it, letting it run faster on cheaper hardware. It happens after training is finished; you are not retraining the model, you are compressing the finished thing. And it pairs naturally with the last two chapters: quantize a distilled small model and you get something genuinely tiny that still holds its own.

The catch is only that rounding is lossy. Round too gently and the model is smaller with no real cost. Round too hard and the coffee finally does start to taste off, the answers degrade. The craft is finding how coarse you can go before quality slips, and modern methods push that line remarkably far. It is the unglamorous final step that takes a capable model and makes it small and cheap enough to actually put in your pocket, closing the loop this book opened at the Blackbird counter.

PLATE XX · ROUNDER NUMBERS, SAME COFFEE

16-bit · heavy

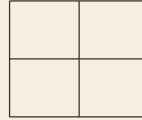


37.412 ml

round



4-bit · ¼ the size



≈ a splash

Each number is stored more coarsely, quartering the model's size. Done with care, the answers barely change, like rounding a recipe no one can taste.

IN ONE SENTENCE

Quantization shrinks a finished model by storing its numbers at lower precision, making it smaller and faster at a small, controllable cost to quality.

One counter, twenty words

You started at the Blackbird counter with a new hire who had read almost everything ever written, and you end it holding the whole vocabulary. A large language model predicts the next token. It reads in pieces, holds meaning as position, and pays attention across a sentence. It taught itself by covering words and guessing them, inside the architecture that made all of this scale. We gave it a job, showed it examples, let it look things up, reach other systems, act in a loop, and learn from what earns thanks. We taught it to think before it speaks, to see and hear, and finally to shrink small enough to carry.

Every headline you meet from here is a rearrangement of these twenty words. When the next one lands, you will not be nodding along. You will know exactly which part of the shop it is talking about.

Prithvi Datla

KILOBYTE COLLECTIVE

PLATE XXI · THE WHOLE VOCABULARY

 LLM <i>predict the next word</i>	 TOKENIZATION <i>words chopped into pieces</i>	 EMBEDDINGS <i>like meanings sit close</i>	 ATTENTION <i>which words change which</i>	 SELF-SUPERVISED <i>cover a word, guess it</i>
 TRANSFORMER <i>the engine it all runs on</i>	 FINE-TUNING <i>learning the house style</i>	 FEN-SHOT <i>show, don't explain</i>	 RAG <i>look it up, then answer</i>	 VECTOR DB <i>the bridge used to find</i>
 MCP <i>one socket for every tool</i>	 CONTEXT ENG. <i>set the context first</i>	 AGENTS <i>doesn't answer, acts</i>	 RL / RLHF <i>learning what seems handy</i>	 CHAIN OF THOUGHT <i>one step at a time</i>
 REASONING <i>thinks first, then plans</i>	 MULTIMODAL <i>words, notes, pictures</i>	 SMALL MODELS <i>the ship aboard to a cart</i>	 DISTILLATION <i>the teacher's work, cheaper</i>	 QUANTIZATION <i>less precise, much lighter</i>

Every word in this book is somewhere in this room. You already understand the room.

Plate XXI · Twenty words, twenty pictures. The whole book on one plate.

SPECIMEN PLATE · THE WHOLE VOCABULARY AT A GLANCE

One last thing.

If you're building with AI and feeling lost, that's the normal part, not the embarrassing part. It is literally what we do at Kilobyte Collective: we build the thing, and we prove it works, with acceptance tests written into the contract.

If that's you, we'd love to help.

[Tell us what you're building →](#)

Tell us what you're building and where it's stuck. A real person here reads every one.