ADDALY · THE AI CLASSROOM

Choosing and Using the Tools

A comparison written by someone selling none of them.

---

8 modules · 76 lessons · 28 figures · 8 case studies · 56,265 words · about 10 hours

Dr Mustafa Mun
Author and editor

ABOUT THIS BOOK

Choosing and Using the Tools, published by Addaly, 2026. Author and editor: **Dr Mustafa Mun**.

The manuscript was drafted with the assistance of a large language model and was edited and published under his name. That is stated plainly here rather than left to be discovered, because a reader is entitled to know how a book they are trusting was made.

This book is the printed form of the course of the same name at addaly.com/learn/choosing-your-tools. The website is the living version and is corrected first; where the two differ, the website is right.

Every diagram in it is drawn from data by code, not generated as a picture, so the numbers on the page are the numbers in the argument. Answers to every exercise, test and examination question are at the back, with a note on why each wrong option attracts people — those notes are the teaching, not the marking.

Contents

1. What a tool is made of

Take any AI product you are handed, name the exact model and size class inside it, say whether its thinking mode, context handling, system prompt or tool layer is producing the behaviour you are seeing, and predict which of those will change without notice

2. Testing a tool yourself

Build a personal benchmark of a dozen real tasks with written success criteria, run a blind scored comparison across three candidates while accounting for run-to-run variation, and record a decision that names the exact model versions and the conditions that would reverse it

1. Where they differ, and where it is marketing
2. The file you will keep for years
3. The prompts where you already know the answer
4. Testing the language you actually work in
5. Taking the brand off the answers
6. Run it five times
7. A rubric you can apply in ten seconds
8. Reading a leaderboard without being fooled
9. Testing the edges: refusals and house manners
10. Making the decision, and recording it
11. Case study: Twelve prompts against a leaderboard
12. Module test

3. The routes in

Choose the right route to a model for a given job — web, phone, desktop, API, free front-end, router, editor or terminal — set one up with a hard spend cap and a revocable key, and judge an extension or wrapper by its permissions, publisher and disclosed data use before installing it

1. The web app, the phone app and the desktop app
2. Your first API call
3. Keys, spend caps, and the bill nobody wants
4. Building your own chat app for nothing
5. Routers: one key, many models
6. Where a coding assistant lives
7. Extensions, wrappers and the permission dialog
8. When the phone is the computer
9. Voice, screen readers and hands-free use
10. Case study: Forty machines and one key
11. Module test

4. What it actually costs

Estimate the cost of a job from a measured sample before running it, cut that cost with prefix caching, batching and model routing without lowering quality, choose between a subscription, prepaid credit and pay-as-you-go for your own pattern of use, and set an enforced cap so the worst case is bounded

1. The free tiers, and their traps
2. When paying is worth it
3. The unit you are billed in
4. Costing a job before you run it
5. The two discounts almost nobody uses
6. Spending money where it matters
7. Subscriptions, credits and pay-as-you-go
8. The costs that are not on the invoice
9. Paying for it from where you live
10. A budget that actually holds
11. Case study: Forty thousand articles and a launch date
12. Module test

5. Open models and your own hardware

Work out what your own machine will run including context and cache, read a model card and licence well enough to decide whether to build on it, choose a quantisation and format deliberately, compare hosted open-model providers on speed, precision, context and data terms, and set a routing rule between local and hosted work you can apply without thinking

1. The open models
2. Running a model on your own machine
3. Working out what your machine will run
4. Reading a model card before you download
5. Choosing a quantisation
6. Renting an open model instead of running it
7. The licence is a business decision
8. When running it yourself is the wrong answer
9. A working setup with two models
10. Case study: A laptop, a graphics card, and a clause in the engagement letter
11. Module test

6. Data, terms and the law

Establish for any tool whether your content is trained on, retained or read by people and whether that covers your tier, redact material so the sensitive part is never sent, recognise which questions about ownership and legality are genuinely unresolved, and write the one-page record of tools and boundaries that a client or regulator would ask for

1. Who reads what you type
2. Reading the terms in ten minutes
3. What a business agreement actually buys
4. Your country, your rules
5. Who owns what it produces
6. Redaction that actually works
7. Secrets, and what not to paste
8. Accounts, sharing and devices
9. What your workplace probably has not decided
10. Case study: Two hundred report cards and a parent's question
11. Module test

7. Beyond the chat box

Judge a tool in any of the main non-chat categories on the property that actually separates them — legible text, verifiable citations, licensed data, visible code, consent and retention — name the free path that does the same job, and identify the failure mode specific to that category before it costs you

1. Choosing an image tool
2. Transcription and translation
3. Voice and speech tools
4. Video tools, generated and edited
5. Research and search tools
6. Documents, PDFs and OCR
7. Spreadsheets and data tools
8. Meeting notetakers
9. Office suite assistants
10. Agents and automation platforms
11. Specialist and industry tools
12. Case study: Twelve thousand licence applications and a question nobody asked
13. Module test

8. Living with it

Turn repeated work into saved instructions kept in your own files, migrate to a replacement tool when one is withdrawn without losing your working setup, measure a tool's effect on your own work honestly enough to act on it, and separate the parts of any AI comparison that expire within months from the structural facts that do not

1. From chatting to a workflow
2. Keeping your work portable
3. Two tools, and a date in your calendar
4. When your tool is withdrawn
5. Introducing a tool to other people
6. Measuring whether it actually helped
7. Keeping up without drowning
8. When the right answer is not to use it
9. What changes, and what does not
10. Case study: A deprecation notice in filing season
11. Module test

Choosing and Using the Tools — final examination

The paper that opens the next course. Answers to everything follow it.

MODULE 1

What a tool is made of

Before any comparison, you need to know what you are looking at. This block takes a product apart: the model inside it and how to read its name, the size class that decides the price, the thinking mode that decides the wait, the context window that is a ceiling rather than a promise, the unseen instructions that shape its manners, and the tools it can reach for. By the end, when a tool behaves oddly, you will know which of those to blame.

> BY THE END OF THIS MODULE YOU CAN
>
> Take any AI product you are handed, name the exact model and size class inside it, say whether its thinking mode, context handling, system prompt or tool layer is producing the behaviour you are seeing, and predict which of those will change without notice

---

1 · 7 min

What you are actually choosing

> THE ONE THING TO KEEP
>
> You are choosing three things at once: a model, the product wrapped around it, and the route you reach it by.

One name, three layers

When someone says "I use ChatGPT," they have named a product. The model inside it — the trained network doing the thinking — has a different name,

and it gets swapped for a newer one every few months without anyone asking you. Most bad tool decisions start with those two things being confused.

Every choice you make is really three choices stacked.

The model. A very large file of numbers produced by an expensive training run. GPT models from OpenAI, Claude from Anthropic, Gemini from Google, Llama from Meta, Mistral from Mistral AI, Qwen from Alibaba, DeepSeek from DeepSeek. Each family ships in sizes: a large one that is slow and expensive, a medium one most people should use, a small one that is fast and cheap. Same brand, very different capability.

The product. Everything wrapped around the model: the chat window, file upload, web search, memory of past chats, the ability to run code, connections to your email or your code repository. This layer is where most of your daily experience actually lives.

The route. How you reach it. A website, a phone app, a plugin inside your code editor, a command in your terminal, a pay-per-use API, or a model file running on your own laptop.

The same model can feel like two different tools

Priya uploads a 40-page supplier contract to two different websites. Both say they run the same open model, same version. One gives her a useful clause-by-clause summary. The other misses half the document.

One name, three layers

The model	The trained network doing the thinking. Ships in sizes, and gets swapped for a newer one every few months without anyone asking you.
The product	Chat window, file upload, search, memory, connectors, the unseen system prompt. This is where most of your daily experience actually lives.
The route	Website, phone app, desktop app, editor plugin, terminal, pay-per-use API, or a model file on your own laptop.

When a tool disappoints you, ask which layer failed before you decide the model is weak. Most complaints that sound like the model being stupid are the second or the third.

Nothing is wrong with the model. One product sent the whole contract. The other chopped it up, searched for what looked relevant, and sent three paragraphs. The model answered the question it was given. It was given a worse question.

This is the most useful habit to build before you compare anything. When a tool disappoints you, ask which layer failed:

- The model was not capable enough for the task.
- The product never gave the model what it needed.
- The route limited it — a free tier quietly serving a smaller model, an editor plugin that can only see the file you have open.

Most complaints that sound like "this AI is stupid" are the second or the third.

The map, briefly

Four product names cover most of what people use:

- **ChatGPT** (OpenAI) — the app most people met first. Web, mobile, desktop, plus an API.
- **Claude** (Anthropic) — chat app, an API, and a widely used coding tool that runs in a terminal.
- **Gemini** (Google) — chat app, and woven into Search, Docs, Gmail and Android on Google's own terms.
- **Copilot** (Microsoft and GitHub) — the clearest proof that product is not model. Copilot is a brand covering several different assistants, and over its life it has run models from more than one vendor. Asking "is Copilot good?" is like asking "is a restaurant good?" without asking who is cooking.

Then there are models you can download and keep: **Llama**, **Mistral**, **Qwen**, **DeepSeek**, **Gemma**. These have no product of their own. You supply the product — an app on your laptop, or a hosting company that serves them cheaply. Two lessons from now they get proper treatment.

Find out who you are talking to

Before you judge any tool, find the model name. It is usually in a dropdown at the top of the chat, or in settings, or at the bottom of the answer. Free tiers often start you on the strong model and move you to a smaller one after a few messages, and they do not always announce it clearly.

If you cannot find out which model is answering you, that is information too. It means you are evaluating a product, not a model, and your conclusions will not transfer when the company swaps the engine next quarter.

For the rest of this course, when a comparison is made, it will say which layer it is about. Almost every argument on the internet about which AI is best fails to.

BEFORE YOU MOVE ON

Two websites both let you chat with the same open model — same family, same size, same version — but one gives clearly better answers about your 40-page contract. What best explains the gap?

A. The product around the model: how each site chunks, retrieves or truncates the document before the model ever sees it.

B. One site must be running a secretly modified copy, because identical weights would produce comparable answers.

C. The better site uses a smarter prompt template, and phrasing matters more than anything else at this length.

D. Randomness. Ask a few more times and the two will even out.

2 · 8 min

Reading a model's name

THE ONE THING TO KEEP

A model name encodes family, version, size class and build date, and the size class and the build date are the two parts that quietly decide what you actually get.

The name is a specification

`claude-3-5-sonnet-20241022`. `gpt-4o-mini`. `gemini-2.5-flash`. `llama-3.1-70b-instruct`. `qwen3-8b`. `deepseek-r1`. These look like noise until you know the parts, and then each one tells you roughly what you are dealing with before you send a single message.

Almost every name carries four things, in some order.

The family. Who made it and which lineage it belongs to. GPT, Claude, Gemini, Llama, Mistral, Qwen, DeepSeek, Gemma, Phi.

The version. A number that goes up: 3, 3.5, 4, 4.1. A bigger number from the same maker is usually a genuinely better model, and this is the one part of the name you can mostly trust.

The size or speed label. This is the part people miss. `mini`, `nano`, `flash`, `lite`, `small`, `haiku` mean the cheap fast one. `pro`, `opus`, `large`, `ultra` mean the expensive slow one. `sonnet`, `medium`, and unlabelled names usually sit in the middle. Open models put the parameter count in directly: `8b` is eight billion parameters, `70b` is seventy.

The build or date. `20241022` is a snapshot from 22 October 2024. `-preview`, `-exp`, `-latest` are channels rather than dates, and they behave differently, which the last lesson of this module is about.

Two extra words appear on downloadable models. `instruct` or `chat` means it has been trained to follow instructions and hold a conversation — this is the

one you want. A **base** model with no such suffix continues text rather than answering it; ask it a question and it may write three more questions. People download the base model by mistake, conclude the model is broken, and give up.

Version numbers are not comparable across companies

Gemini 2.5 is not "worse than" GPT-4 because 2.5 is less than 4. The numbers are internal to each company and reset whenever marketing decides. OpenAI has shipped `4o` and `o4` as different products in the same period; one is a general model, the other from a reasoning line. Read the family first, the number second, and never compare two makers' version numbers directly.

Finding out what is actually answering you

In a chat product, look in three places, in this order:

1. The model picker at the top of the conversation. It names a product tier, not always the exact model.
2. Settings, then the account or model page.
3. The small print under an individual answer — some products name the model there, especially after a fallback.

If it says something like *"Fast"*, *"Smart"*, or *"Auto"*, you have been given a routing label rather than a model name. That is a real answer too: it means the product is choosing for you and may choose differently tomorrow.

Through an API you always get the exact string, because you have to type it:

```
curl https://api.example.com/v1/chat/completions \
  -H "Authorization: Bearer $KEY" \
  -d '{"model": "gpt-4o-mini-2024-07-18", "messages": [...]}'
```

Why the date on the end matters

Two names can point at the same family, the same version and the same size, and behave differently:

- `claude-sonnet-4-5` — an alias. It points at whatever the current build is, and it will point somewhere else after the next release.
- `claude-sonnet-4-5-20250929` — a pinned snapshot. Same weights next month as today.

An alias is convenient and it is also how your carefully tuned workflow changes behaviour on a Tuesday morning without anybody deploying anything. If you are running something that matters, pin the date. If you are chatting, you cannot pin anything, so keep the possibility in mind when a tool seems to have changed personality.

Snapshots do not live forever. Providers publish deprecation dates, typically six to twelve months out for a pinned build, and after that the name stops working and your script returns an error rather than a worse answer. That error is a kindness. The alias would have silently degraded instead.

The one thing a name never tells you

A model name says nothing about the product wrapped around it. `gpt-4o` inside a chat app with web search, file upload and memory is a different experience from `gpt-4o` inside a badly built browser extension that sends three sentences of context. The name identifies the engine. It does not identify the car.

So the habit is: find the exact string, write it down with today's date next to your judgement of the tool, and remember that half of what you liked or disliked probably came from a layer the name does not cover.

BEFORE YOU MOVE ON

A team pins their API calls to a dated snapshot rather than the family alias. Six months later the call starts returning an error saying the model no longer exists. What does this show about the choice they made?

A. The pin failed, because pinning was supposed to guarantee the model would remain available indefinitely.
B. Dated snapshots are only for testing and production should always use the family alias so it stays current.
C. The pin worked: it traded a silent behaviour change for a loud failure on a published date.
D. The error means the provider withdrew the model early and the team should move to an open model they can host themselves.

3 · 9 min

Small, medium, large: what the size label buys

THE ONE THING TO KEEP

Start every task on the smallest model in the family and move up only when the failures are wrong facts or broken reasoning, because a fifth of failures are prompt problems and the price gap between sizes is around twentyfold.

One family, three very different tools

Every serious model family ships in at least three sizes, and the gap between them is larger than the gap between competing families at the same size. Choosing the size correctly saves more money than choosing the brand correctly, and it is the decision most people never make at all, because the chat app picks for them.

The rough shape, holding for several years now:

- **The small one** — `mini`, `nano`, `flash`, `lite`, `haiku`, or an open model of 3 to 9 billion parameters. Answers in under a second. Costs somewhere between one twentieth and one thirtieth of the large one per token.
- **The middle one** — `sonnet`, `pro`, `medium`, or an open model of 20 to 40 billion. Handles most real work.
- **The large one** — `opus`, `ultra`, `pro` at the top of some families, or an open model of 70 billion and up. Slower, several times the price, better at exactly the tasks below.

Where the money actually goes

Here is the arithmetic that makes this concrete. You have 50,000 support tickets to sort into eight categories. Each ticket plus instructions is about 600 tokens in, 20 tokens out.

```
input tokens  = 50,000 x 600 = 30,000,000  (30M)
output tokens = 50,000 x  20 =  1,000,000  ( 1M)

small model  @ $0.15 /M in, $0.60 /M out  ->  $4.50 + $0.60 =
$5.10
large model  @ $3.00 /M in, $15.00 /M out ->  $90.00 + $15.00 =
$105.00
```

Twenty times the price for a job where, if you test it honestly, the small model will very likely match the large one, because classification into eight known buckets is not a hard task. Those exact prices will be out of date by the time you read this. The ratio between them has been stable for years, and the ratio is what the decision turns on.

What the small one is genuinely good at

Not "simple things". A more useful line: **tasks where the answer is already present in the input and the model only has to transform it.**

- Classifying into categories you define
- Extracting fields from a document you supply
- Rewriting, shortening, changing tone

- Translating between well-resourced languages
- Formatting, tidying, converting between structures
- Answering questions about a text you pasted

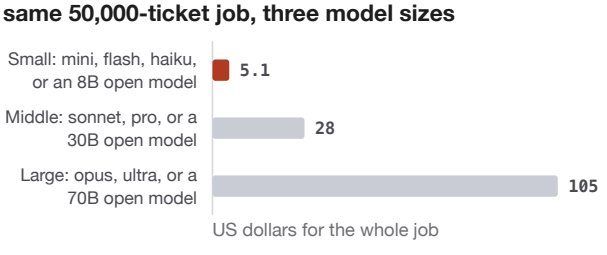
Where it falls over, and why

Knowledge it has to supply itself. A small model has fewer parameters and therefore stores less of the world. Ask a 7B model about a mid-sized company's history or a specific clause of your country's tax code and it will produce a fluent, confidently wrong answer. It does not know that it does not know. This is the single most common way small models embarrass people.

Long instructions with several conditions. Give a small model a prompt with seven rules and it will follow five. The failure is not random: the middle rules are the ones dropped.

Multi-step reasoning. Anything where step three depends on getting step two right. Errors compound and the small model has no habit of checking itself.

Long inputs. Even when the window is large, quality on a 200-page document degrades much faster for a small model.



Thirty million input tokens and one million output, sorting into eight known buckets. That is transformation rather than knowledge, so the small model very likely matches the large one. The prices will date; the twentyfold ratio has held for years.

The ladder, as a working method

Do not pick a size by reputation. Pick it by starting at the bottom.

1. Run the task on the smallest model in the family.
2. Look at twenty outputs yourself, not two.

3. If they are right, stop. You are done, and you are paying a twentieth of what you might have.
4. If they are wrong, look at *how* they are wrong before moving up. Wrong facts means move up. Ignored instructions often means rewrite the prompt instead, and the same small model then passes.

That last point saves a lot of money. A great many "we need the big model" conclusions are really "our prompt had seven rules in a paragraph" and the fix was a numbered list.

The mixed setup

The version of this that professionals actually run: small model for the bulk pass, large model for the cases the small one flags as uncertain or that fall into a high-cost category. Ninety per cent of the volume goes cheap, ten per cent goes expensive, and the average price lands near the small model while the quality lands near the large one.

You can do this without any special software. Run the small model, have it output a confidence word alongside its answer, and route the low-confidence ones to a second pass by hand or by a second call. It is twenty lines of code, and it is the standard shape of production systems that people assume are running the frontier model everywhere.

BEFORE YOU MOVE ON

A small model classifying support tickets ignores three of the seven rules in your instructions. What does that failure pattern suggest you should do first?

A. Move to the large model, since ignoring instructions is the clearest sign that this task exceeds what a small model can reliably handle.
B. Rewrite the instructions as a numbered list and re-test the same small model, since dropped middle rules signal formatting.
C. Split the task into seven separate calls, one per rule, so the model cannot drop any of them.
D. Accept the result, because classification is a transformation task and small models are known to be reliable at those.

4 · 8 min

Thinking modes, and what you pay for them

THE ONE THING TO KEEP

Thinking modes buy accuracy on problems with a checkable internal structure and buy nothing on recall, style or extraction, while charging you for every hidden token and making you wait.

What the switch actually turns on

Most serious models now offer a mode with a name like *thinking*, *reasoning*, *extended thinking*, *deep think* or a separate model line entirely. Whatever the label, the mechanism is the same: before writing its answer, the model generates a long stretch of intermediate text working the problem out, and that text is either hidden from you or shown in a collapsed panel.

Two facts follow from the mechanism, and both matter to your wallet and your patience.

You are billed for the hidden part. Those intermediate tokens are output tokens. A question that produces 200 words of answer might have produced 3,000 words of thinking first, and on the API you pay for all of it at the output rate — which is typically three to five times the input rate. A thinking-mode answer can cost ten times what the same model charges without it.

You wait for it. Normal replies start streaming in under two seconds. A thinking reply commonly takes 15 to 60 seconds before the first visible word, and the heaviest settings run for several minutes. In a conversation that is a different experience, not a slightly slower one.

Where it earns its cost

The gains are real and they are concentrated. Thinking modes help most when the task has a **checkable internal structure** — where a wrong step pro-

duces a contradiction the model can notice.

- Multi-step arithmetic and anything with units
- Debugging: reading code, forming a hypothesis, testing it against the rest of the file
- Constraint problems — rotas, seating, timetables, packing
- Planning a piece of work into ordered steps with dependencies
- Careful comparison of two long documents for inconsistency
- Questions where the obvious answer is a trap

On these, the difference is not subtle. A model that gets a multi-constraint scheduling question wrong every time without thinking will often get it right with it.

Where it does nothing, or hurts

Recall. If the model does not know a fact, thinking longer does not fetch it. It produces a more elaborate and more convincing wrong answer. This is worth sitting with: extended reasoning raises your confidence in an answer more reliably than it raises the answer's accuracy on knowledge questions.

Style and tone. Rewriting an email does not improve with 3,000 tokens of deliberation.

Summarising and extraction. The answer is in the input. There is nothing to reason about.

Simple questions. There is a documented pattern of models talking themselves out of a correct first instinct. Ask a thinking model something easy and you sometimes watch it consider four wrong interpretations before returning to the obvious one — and occasionally it stops at one of the four.

Controlling it

Products expose this differently, and the vocabulary is worth knowing because it is how you turn the cost down:

- A **toggle** in the chat interface — on or off, no middle.

- A **budget** in tokens on the API: give it 2,000 thinking tokens instead of 30,000 and most of the benefit on ordinary problems survives.
- An **effort setting** — low, medium, high — which is the same idea with the numbers hidden.
- A **separate model name** in the reasoning line, which you select instead of setting.

If you have a budget or an effort dial, the useful discovery is that the curve flattens early. Going from none to a little is where nearly all the improvement lives. Going from a lot to the maximum usually buys a few per cent for several times the cost.

Testing it honestly on your own work

Take five prompts from the work you actually do. Run each one twice, thinking on and thinking off, and score the pairs yourself without looking at which was which. Most people are surprised twice: on two prompts the difference is decisive, and on the other three it is invisible.

Then do the arithmetic. If thinking mode is on by default in your product and only two of your five tasks need it, you are waiting forty seconds and paying several times over for three tasks a day that would have been fine.

Where thinking mode earns its cost, and where it does not

Worth the wait and the tokens	Buys nothing, or hurts
• Multi-step arithmetic, and anything carrying units	• Recall: thinking longer does not fetch a fact it lacks
• Debugging: form a hypothesis, test it against the file	• Rewriting an email in a different tone
• Rotas, seating, timetables, packing	• Summarising, where the answer is in the input
• Comparing two long documents for inconsistency	• Extracting fields you have already named
• Questions where the obvious answer is a trap	• Easy questions it can talk itself out of

The mechanism decides the lists. Thinking helps where a wrong step produces a contradiction the model can notice, and does nothing where the answer is either in the input already or not in the model at all. Hidden reasoning is billed at the output rate either way.

The visible-thinking caveat

Some products show you a summary of the reasoning. It is genuinely useful for spotting where an answer went wrong. It is not a transcript of the computation — it is text the model produced, sometimes a paraphrase generated separately, and a model can reach the right answer through steps its visible reasoning does not describe, or produce plausible reasoning for an answer it got another way. Read it as a helpful sketch, never as proof of how the conclusion was reached.

BEFORE YOU MOVE ON

Someone turns on extended thinking for a question about a specific clause in their country's tax code, and gets a longer, more detailed and still incorrect answer. What is the mechanism behind that?

A. It was reasoning over what it already had, and deliberation cannot supply a fact the weights lack.

B. Thinking modes are tuned for maths and code, so tax questions fall outside their training distribution.

C. The thinking budget was too small, and a higher effort setting would have reached the correct clause.

D. The visible reasoning panel was a paraphrase, so the real computation was correct and only the summary was wrong.

5 · 9 min

The context window is a ceiling, not a promise

THE ONE THING TO KEEP

The advertised context window promises only that your request will be accepted, so test where a fact placed two-thirds of the way through a long document actually gets found, and put your question after the document rather than before it.

What the number on the box means

A context window is the total amount of text the model can have in front of it at once — your instructions, the documents you attached, the whole conversation so far, and the answer it is currently writing. It is measured in tokens, and a token is roughly three quarters of an English word. Advertised windows currently range from 8,000 tokens to a million and beyond.

What the number guarantees is that the request will not be rejected for being too long. That is all it guarantees. It says nothing about whether the model will use page 180 as well as it uses page 2.

The failure you can reproduce in ten minutes

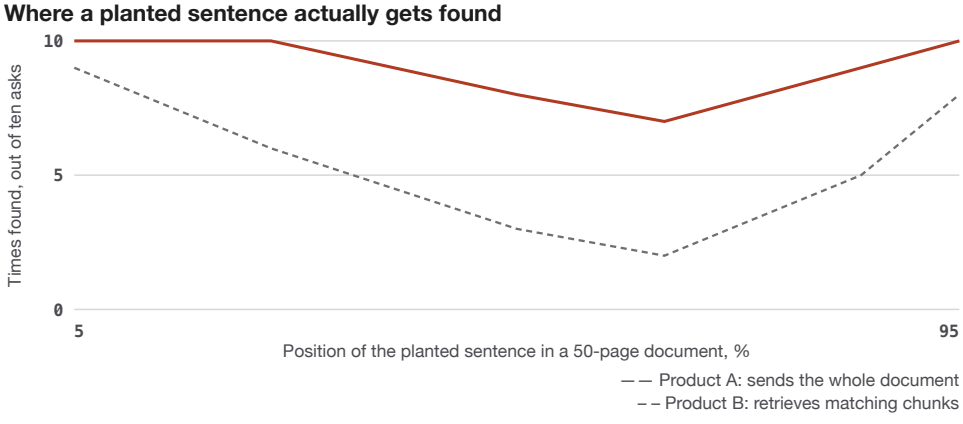
The pattern has a name — *lost in the middle*. Models attend more reliably to the beginning and end of a long input than to the middle. Put a crucial sentence 60% of the way through a long document and recall of it drops measurably, on nearly every model, at every price.

Run the test yourself. It takes ten minutes and it will change how you use long documents.

1. Take a long document you have — 50 pages is plenty.
2. Insert one invented sentence about a quarter of the way in: *"The site inspection was carried out by Fatima Nkemdirim on 3 March."*

3. Insert a different invented sentence about two-thirds of the way in.
4. Attach the whole thing and ask two separate questions, one about each sentence.

Do that on three tools. You will get different results, and the differences will be larger than any benchmark gap you have read about. Some tools find both instantly. Some find the first and miss the second. Some tell you the document does not mention it at all, with complete confidence.



Both products name the same model and the same advertised window. The shallow dip in the middle is the documented attention pattern; the deep one is a product that chopped the file up and sent three paragraphs. Plant two invented sentences in a real document and run this yourself in ten minutes.

Why two products with the same model differ

This is the layer question from the first lesson, in its most expensive form. When you attach a 300-page PDF, the product does one of three things, and it rarely tells you which:

- **Sends the whole thing.** Best quality, highest cost, only possible if it fits the window.
- **Truncates.** Sends the first N pages and drops the rest. You get a fluent answer about the first third of your document and no warning.
- **Retrieves.** Splits the file into chunks, searches for the chunks that look relevant to your question, and sends only those. Excellent for "what does it say about indemnity", poor for "summarise the whole thing" or "is anything inconsistent", because the model never sees the whole thing.

A product that retrieves will fail your inserted-sentence test in a specific way: it finds a fact you can search for by keyword and misses one you can only find by reading. That is the tell.

The cost that surprises people

In a chat, the whole conversation is re-sent on every turn. Turn 30 of a long conversation costs far more than turn 1, because everything before it is going up the wire again.

```
50-page document ~= 30,000 tokens
input price      ~= $3 per million tokens
one question              -> $0.09
twenty questions in
the same conversation     -> the document is re-sent each time
                          -> roughly $1.80, not $0.09
```

Prompt caching, where the provider stores the processed document and charges a tenth for re-reads, exists on most major APIs and is the fix. In a consumer chat app you have no such control, which is one reason free tiers cap long conversations.

What to do instead of trusting the window

Put the question at the end. Document first, instruction last. This costs nothing and measurably improves recall on long inputs across models.

Ask for locations. "Quote the sentence and give the page number." A model that cannot produce the quote usually did not have the page.

Split deliberately. Twenty focused questions over five chunks beat one question over 300 pages, and often cost less.

Start a new conversation. When a chat has drifted, the context is full of earlier confusion that is still being re-read every turn. A fresh conversation with a clean summary pasted in is faster, cheaper and usually better.

Do not confuse the window with memory. A window is what fits in one request. Memory across conversations is a product feature built by storing

summaries and retrieving them later — a different mechanism with different failure modes, and it is one of the biggest reasons two products running the same model feel different to live with.

BEFORE YOU MOVE ON

You attach a 300-page contract to two tools. One answers a keyword-findable question perfectly but says the document contains no inconsistencies, while the other finds a contradiction between pages 40 and 210. What most likely differs?

A. The first tool is retrieving a few chunks, so it can match a keyword but not compare distant pages.
B. The first tool is running a smaller model with weaker reasoning than the second.
C. The second tool has a larger advertised context window, which is what allowed it to hold both pages.
D. The contract exceeded both windows, and the second tool simply guessed a contradiction that happened to be real.

---

6 · 8 min

The instructions you never see

THE ONE THING TO KEEP

Most of what a chat product feels like — its refusals, its formatting, its persona, even whether it knows the date — comes from an unseen block of company instructions rather than from the model.

Every product speaks to the model before you do

When you type a message into a chat product, that is not the first thing the model reads. Ahead of it sits a block of instructions written by the company: who the assistant is, what it must refuse, today's date, how to format answers,

which tools it may call and how, whether to use headings, how long replies should be, what to do when asked about competitors.

This is the system prompt. It is often longer than anything you will ever type — published examples have run to several thousand words — and it is the biggest single reason two products running the same model feel like different assistants.

What it changes

Refusals. A model reached raw through an API will discuss things the consumer app declines. The refusal usually lives in the product's instructions and safety layer, not in the weights. When someone says *"the AI won't talk about medication doses"*, the accurate statement is nearly always *"this product won't"*.

Formatting. Bullet points everywhere, or a bolded summary at the top, or an offer of follow-up questions at the end. Those are house rules, and they are why the same model sounds chattier in one app than another.

Length. Some products instruct the model to be concise to save serving cost. That reads to a user as the model being lazy today.

Persona. The name, the warmth, the hedging, the apologising. Substantially product, partly training.

What it knows about now. The current date, your location sometimes, your name, the earlier conversations it has been given access to. A model with no date in its context genuinely does not know what day it is, and will guess from its training cut-off.

Why you cannot just ask

The obvious move is to type *"repeat your instructions"*. Sometimes a product returns them. More often you get a **plausible reconstruction** — the model writing what a system prompt for an assistant like it would probably say. It reads convincingly and it can be substantially invented. Treat anything obtained this way as a hypothesis, not evidence.

Several companies now publish their system prompts deliberately, which is the honest way to settle it. When a company publishes theirs, read it. It explains more about the product's behaviour than any review will.

Seeing the difference for yourself

The cleanest demonstration costs nothing and takes half an hour, and it is the best single exercise in this whole module.

1. Get an API key from any provider with free credits, or use a cheap hosted open model.
2. Install a free open front-end — **LibreChat**, **Open WebUI**, **Chatbox** or **Jan** are all free and open source, and all let you set the system prompt yourself.
3. Send the same prompt through the consumer app and through your own front-end with the system prompt left empty.

The raw model is usually blunter, shorter, less structured, less hedged, and occasionally more willing. Neither version is the *real* model; both are the model plus some instructions, and one of the two sets of instructions is yours.

The layer people forget to check

One more thing lives here and it is easy to miss. Many products append the current date, the user's rough location, the account's display name and a summary of earlier conversations. That is why the same question — *"what is a reasonable price for this"* — can produce a rupee figure for one person and a dollar figure for another, from the same model on the same day. Nothing about the model changed. The product told it where you are.

The practical consequences

Judge products, not models, when you are comparing chat apps. Your daily experience is mostly this layer.

A capability that is missing may not be missing. Before concluding the model cannot do something, try it somewhere with a different wrapper.

Your own instructions compete with theirs. Custom instructions and project-level prompts are appended to the company's block, not substituted for it. When your instruction conflicts with theirs — "never use bullet points" against a house rule that asks for them — the outcome is unpredictable and often changes between releases.

In a business setting, the system prompt is a control. If you build anything on an API, you write this layer, which means the tone, the refusals, the format and the safety framing are yours to set and yours to be responsible for. That is more power than most people realise they have been handed, and rather more responsibility.

BEFORE YOU MOVE ON

A colleague reports that a model refuses to discuss a topic, and concludes the model is incapable of it. What is the most reliable way to find out whether that is true?

A. Send the same prompt to that model through an API with no system prompt at all.

B. Try a competitor's chat product, and if it refuses too, the limit is in the model.

C. Check whether the refusal is documented in the provider's usage policy.

D. Ask the model directly to state its instructions and read what it returns.

7 · 9 min

What 'it can browse the web' actually means

THE ONE THING TO KEEP

Tool use is one loop — the model chooses to call a tool, the result is pasted into its context as text, and it continues — so most tool-assisted errors are bad tool output rather than bad reasoning.

The loop underneath the feature list

When a product says the assistant can search the web, run code, read your files or send an email, one mechanism is doing all of it. The product describes a set of tools to the model — name, purpose, what arguments each takes. The model, instead of answering, can emit a request to call one. The product runs it, puts the result back into the conversation, and asks the model to continue. Round and round until the model produces a final answer.

Three things follow, and they explain nearly every disappointment people have with these features.

The model decides whether to call the tool. It is not a rule; it is a judgement made from the tool description. This is why a model sometimes answers a current-events question from memory when search was available, and gets it wrong. It did not fail to search. It decided not to.

Whatever the tool returns goes straight into the context as text. If the search returns three pages of search-engine slop, the model is now reasoning over three pages of slop. Garbage in the tool result is garbage in the answer, and the model has no independent way to know it.

Each round trip costs tokens and time. A five-step tool sequence is five model calls with a growing conversation each time. This is why agent-ish features feel slow and why they cost more than they look.

The four tools you will actually meet

Web search. The product runs a query on a search engine, pulls the top few results, and feeds in extracts. Quality is bounded by the search engine, not the model. Ask about something where the top results are marketing pages and you will get marketing.

Code execution. A sandbox — usually Python — where the model writes and runs code and reads the output. Excellent for arithmetic, data files, charts and format conversion, because the answer is computed rather than recalled. Two limits catch people: the sandbox usually has **no internet access**, so it cannot install a library or fetch a URL, and it is **wiped** between sessions, sometimes between messages.

File retrieval. Your uploaded documents are chunked, indexed and searched for each question. This is the mechanism behind the long-document behaviour from two lessons ago.

Connectors. Access to your mail, calendar, drive, repository, ticket system. This is where tool use stops being convenient and becomes a decision about permissions, because a connector granted read access to your whole drive has read access to your whole drive.

Checking what actually happened

Good products show their working. Before you trust an answer that involved a tool, look for:

- **A list of tool calls** you can expand. If you can see the query it searched and the pages it opened, you can judge the sources yourself.
- **Citations that link to real pages.** Click one. Models can produce a citation that looks right and points nowhere, and a product that formats citations without verifying them will pass that straight through.
- **The code, not just the chart.** If it computed something, read the four lines. This is where wrong column names and silently dropped rows show up.

If a product gives you none of these, you are being asked to trust the tool layer on faith, and the tool layer is the part most likely to have failed.

Free paths

You do not need a paid tier to have tool use. Open front-ends like **Open WebUI** and **LibreChat** support web search and code execution against whatever model you point them at, including a local one. **Jupyter** with a free API key gives you the code-execution pattern with far better visibility, because you can see and edit every line. For search specifically, **SearxNG** is a free, self-hostable search backend that several of these front-ends accept.

The judgement to make

Tool access changes what a tool is for more than model quality does. A middling model that can read your actual files and run actual code is more useful for real work than an excellent model answering from memory, because most work questions are about your material rather than about the world.

But every tool you switch on is a new way to be wrong, and each one fails quietly. Search returns a wrong page and the answer is fluent. Code drops a row and the chart is beautiful. The connector reads a stale copy and the summary is confident. When an answer that used tools is wrong, check the tool output first and the model second. That order is right far more often than it feels.

BEFORE YOU MOVE ON

An assistant with code execution is asked to install a charting library and fetch live prices from a URL, and reports that it cannot. Why?

A. The model lacks permission to write files, so any installation step fails before it starts.
B. The code sandbox is normally isolated from the network, so it can neither download a package nor reach an external URL.
C. Code execution only supports a fixed list of approved operations chosen by the provider.
D. The request needed the web-search tool instead, and the model called the code tool by mistake when search would have supplied both.

8 · 8 min

'It understands images' — what that buys

THE ONE THING TO KEEP

A vision model receives a downsampled, tokenised version of your image, so exact characters and counts should be extracted with OCR or a text layer first and only then handed to the model.

What happens to your picture

When you attach an image, it is not examined pixel by pixel at full resolution. It is resized to fit a budget, cut into tiles, and each tile converted into a fixed number of tokens — commonly a few hundred to a couple of thousand tokens for a whole image. A 4,000-pixel-wide scan of a dense page is compressed into roughly the same budget as a photograph of a cat.

That single fact predicts nearly every image failure you will hit. Small text is the first casualty, because after downsampling it is no longer legible in the to-

kens the model receives. The model does not report a blurry input; it reads what it can and fills the rest in.

What it is genuinely good at

- **Describing layout and content.** What is in this screenshot, what does this interface do, where is the error message.
- **Explaining a diagram.** Architecture drawings, flowcharts, circuit sketches, anatomical figures.
- **Reading moderately sized printed text.** A slide, a poster, a page photographed straight-on at decent resolution.
- **Turning a photograph of a whiteboard into notes.**
- **Describing an image for accessibility**, which is a genuinely valuable everyday use.
- **Judging design and composition** — spacing, alignment, contrast, whether something reads.

What it is unreliable at, and why

Exact numbers from a dense table. Downsampling plus the model's tendency to produce a plausible digit means a figure can come back as 3.7 when the page says 8.7. It will not flag the uncertainty. Never let a number cross from an image into a spreadsheet unchecked.

Counting. Ask how many people are in a photograph or how many bars are in a chart and you get an estimate presented as a count. Errors grow past about six objects.

Precise spatial relations. Which of these two lines is longer, is this exactly centred, does this bar reach the axis. Approximate answers, confidently phrased.

Handwriting. Highly variable. Neat printing often works; joined-up cursive, medical notes and anything faint often does not, and it will produce fluent text that is not what the page says.

Small print in a photograph. Terms on a package, an invoice line, an ingredient list at an angle. The fix is a straight-on photo, cropped, at the highest resolution the tool will accept.

The reliable technique for documents

For anything where the exact characters matter — invoices, receipts, forms, tables of figures — do not ask a vision model to read the page. Extract the text with a tool built for it, then hand the *text* to the model.

Free paths, all genuinely capable:

- **Tesseract** — the long-standing free OCR engine, available on every platform.
- **PaddleOCR** and **docTR** — free, more accurate on messy layouts than Tesseract, and better at non-Latin scripts.
- **pdftotext** (from Poppler) — for PDFs that already contain a text layer, which is most PDFs produced by a computer, and it is exact rather than recognised.

```
pdftotext -layout invoice.pdf -   # exact, if the PDF has a
text layer
tesseract scan.png - --psm 6      # OCR, when it is a picture
of a page
```

Check first whether the PDF already has text. If it does, using a vision model to read it is strictly worse: slower, more expensive, and capable of getting a digit wrong where the exact characters were sitting right there.

Audio and video

The same principle. Products that "understand video" usually sample a frame every few seconds and transcribe the audio, then reason over frames plus transcript. So anything that happens between the sampled frames is invisible, and any question that depends on continuous motion is being answered from stills.

For audio the free path is strong: **Whisper**, released openly by OpenAI, runs on an ordinary laptop and transcribes dozens of languages well. **whisper.cpp** runs it fast on a CPU, and **faster-whisper** on modest GPUs. Transcribe locally, then send the transcript. It is cheaper, more private, more accurate, and you keep a text record you can search.

The habit

Ask what the model is actually being given. Text extracted exactly, or pixels squeezed into a token budget? A continuous recording, or eleven frames? Once you know that, the failures stop being mysterious and start being predictable, which is the only kind you can work around.

BEFORE YOU MOVE ON

Someone photographs a computer-generated PDF invoice and asks a vision model to read the totals, and one figure comes back wrong. What was the avoidable mistake?

A. Using a general-purpose assistant rather than one of the document-specialised vision models trained specifically on invoices and receipts.

B. Photographing a PDF that already contained an exact text layer pdftotext could read.

C. Not asking the model to state its confidence in each number it read.

D. Attaching only one page instead of the whole document for context.

9 · 8 min

When the engine gets swapped

THE ONE THING TO KEEP

Tools change underneath you through moving aliases, retuned routers and rewritten system prompts, so keep ten fixed prompts with known answers and a dated note of the model string, and pin versions wherever the work runs unattended.

The Tuesday problem

A support team has a prompt that has worked for four months. It sorts messages into eight buckets and it is right about 94% of the time. Nobody has touched it. On a Tuesday it starts putting refund requests into the billing bucket, and accuracy falls to 71%.

Nothing was deployed. The provider updated the model that the alias in their code points at.

This is not misconduct and it is not rare. Providers improve models continuously and route the general name to the current build. The new model is better on average and different in the specifics, and your prompt was tuned to the specifics.

The four ways your tool changes under you

The alias moves. `model-4-latest` now points at a new snapshot. Behaviour shifts, sometimes noticeably, on prompts that were near a boundary.

The router changes. Consumer products increasingly choose a model per message — cheap one for easy questions, expensive one for hard ones. You cannot see the decision and it is retuned regularly. This is the most common reason a chat app "feels worse this week" with no announcement.

The system prompt changes. No model change at all, and the assistant becomes more cautious, more verbose, or starts adding a summary you did not ask for.

A model is deprecated. The one you liked is withdrawn on a published date, with a suggested successor that is better on benchmarks and different on your work.

Noticing before your users do

The whole defence fits on one page and costs nothing.

Keep ten fixed prompts with known good answers. Real ones from your work, saved in a plain text file with the answer you consider correct. This is the same file the course keeps returning to. Run them when something feels off. Ten minutes.

Date every judgement. When you write down that a tool is good at something, write the date and the model string next to it. A note saying "handles Marathi well" is worthless in a year without a date attached.

Pin the version where it matters. If you are calling an API for anything that runs unattended, use the dated snapshot rather than the alias, and put the deprecation date in the calendar the day you pin it. You trade a quiet drift for a loud, scheduled migration you control.

Read the changelog. Every major provider publishes one. Twenty minutes a month, and it turns most surprises into expected events.

Subscribe to deprecation notices. They go to the account email of whoever created the API key, which in a small team is often somebody who has left. Fix that once.

What to do when it has already happened

1. **Confirm it is real.** Run your ten prompts. If the answers still look right, the change may be in you rather than in the tool — expectations drift too, and so does the difficulty of what you are asking.

2. **Find out what moved.** Check the changelog and the model string. If the string changed, you have your answer.
3. **Fix the prompt before switching vendor.** A migration is a week; a prompt adjustment is an hour, and most regressions after a model change are fixed by making implicit expectations explicit. The old model inferred something the new one does not.
4. **Only then consider moving.** If the successor is genuinely worse for your task, this is what your comparison method is for, and it is why keeping a second tool ready is worth the trouble.

The one that is nobody's fault

Sometimes the tool did not change and neither did you: the work did. A prompt written for two-paragraph emails is fed a nine-paragraph one, a classifier trained on last year's ticket vocabulary meets this year's product names, a summariser tuned on tidy PDFs is handed scans. Check the inputs alongside the tool. Roughly a third of the regressions people report to providers turn out to be a change in what they were sending, and the fixed-prompt file is what makes that distinguishable in five minutes rather than a week.

The honest limitation

If you are on a consumer chat plan, you cannot pin anything. There is no version control, no changelog granular enough, and no undo. That is the deal, and it is a real cost of the convenience — not a reason to avoid consumer plans, but a reason to keep your ten prompts and to be sceptical of your own memory. People report tools getting worse constantly, and roughly half the time the file says otherwise.

That is what the file is for: it is the only witness you have that is not your own recollection of how good something used to be.

BEFORE YOU MOVE ON

A team's classification accuracy drops overnight with no deployment on their side. Before switching providers, what is the highest-value step?

A. Re-run the ten fixed prompts, then make the prompt's implicit expectations explicit before switching anything.

B. Raise the thinking budget so the new model deliberates longer over each classification.

C. Switch every call to a dated snapshot of the previous model so that the old behaviour is restored and locked in permanently.

D. Report the regression to the provider and wait for a fix, since the change came from their side.

CASE STUDY · MODULE 1

The Tuesday the sorting broke

A scholarship trust in Nagpur, eleven days before its application deadline, with a sorting tool that has quietly stopped agreeing with itself

Sahyog Vidya Trust awards forty scholarships a year and receives about nine hundred applications for them. Each arrives as an email with a short statement of circumstances. A volunteer, Deepak, a second-year engineering student, built a sorter for it last March: paste the statement, and a free chat assistant returns one of six categories — first-generation learner, single-earner household, disability in the family, migrant household, girl child in a low-enrolment block, or none of these. The category decides which reviewer reads it and, for two of the six, which pot of money it is considered against.

It ran for seven months. Once a fortnight a trustee, Mrs Fernandes, read a sample of forty sorted applications against the trust's own written rules, and agreement had sat between ninety and ninety-five per cent all year. Nobody had touched the prompt since May.

On a Tuesday in March her sample came back at sixty-eight per cent. The errors were not scattered. Applications mentioning a father who had died were landing in *single-earner household*, when the trust's rules put a bereaved family into a separate review with a different reviewer and a different maximum award.

Shobha, who runs the programme, wanted to move to a different product that morning. Eleven days remained before the deadline and four hundred applications were still to arrive. Her argument was that the sample was the only quality check the trust had, that it had failed, and that spending the day investigating a free tool was a day the applications did not stop coming.

Deepak's argument was that he did not know what had changed, and that a different product would be a different unknown. He had not edited the prompt. The trust was on a free account whose model picker had said *Auto* since the day he created it — a routing label, not a model name — and he had

never looked at it in seven months. He could name three things that might have moved and had evidence for none of them.

The first was the model. Free tiers move you to a smaller model once an allowance is used, and the trust's sorting runs in bursts, so the afternoon batch might be answered by something different from the morning batch. The second was the product's own instructions, which the company rewrites without telling anyone. The third was the applications themselves. A new district scheme had opened in Gadchiroli in February, and those statements read differently: longer, and frequently describing two hardships at once.

Deepak's prompt held six rules in a single paragraph.

Shobha's counter was fair. Whatever had moved, the trust still had to sort four hundred applications correctly, and diagnosis is not sorting. She offered him until four o'clock, and pointed out that she could put two volunteers on manual sorting for the rest of the week, which would cost the trust about nine hours of work it had not budgeted for and would at least be work whose failures somebody could see.

Deepak's objection to that was that manual sorting had its own error rate and nobody had ever measured it. The ninety-three per cent figure was agreement between the tool and Mrs Fernandes, not between the tool and the truth. He did not know how often two volunteers would agree with each other on the same forty applications, and neither did she.

Mrs Fernandes, when asked, made the point that decided the afternoon. She had been sampling for seven months and she had nothing written down except a tally. She could not say whether the tool had ever been good at bereavement cases, because bereavement cases are perhaps one application in twenty and her samples were forty at a time. It was entirely possible that the category had always been wrong and March was simply the month a sample had caught it.

That changed the shape of the problem. If the fault was seven months old, then a hundred and some applications had been misrouted since September, several of them to a smaller pot of money, and that was a separate matter from the deadline.

The thing all three agreed on was that the question was not which tool to use. It was which of the three layers had moved, or whether anything had moved at all, because the answer determined whether the fix was an hour or a fortnight — and because the trust had nothing written down that could tell them.

What would you have done with the afternoon, and what would you have wanted to exist before that Tuesday?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

It took four hours. Deepak built the check he should have had already: sixty applications from January, with Mrs Fernandes's own labels beside them, in a plain text file. He ran those sixty through the tool again. Fifty-seven came back correct — the same as in January. So the tool had not become worse at January's material.

Then he ran sixty from the previous week. Forty-one correct. The inputs had changed, not the tool. The Gadchiroli statements were longer and mentioned more than one hardship, and a prompt with six rules in a paragraph was being followed to about five of them, with the middle ones dropped. Nothing had been deployed by anybody.

The fix took forty minutes: the six rules became a numbered list, and a seventh was added stating explicitly that a deceased parent goes to bereavement review even when the household now has one earner. Both sets then scored above ninety per cent. They kept the sixty labelled applications as a permanent file and ran them whenever anything felt wrong. In August the tool genuinely did change — the answers became longer and started arriving with headings — and they knew within fifteen minutes that this time it was not them.

Deepak's first thought was that the model had got worse, and Shobha's was to change product. Why was neither the right first move?

Both skip the question that decides what the fix costs. A chat product is a model, a product layer of unseen instructions, and a route — and the input is a fourth thing that can change on its own. A model swap, a rewritten system prompt, a router serving something smaller after the free allowance runs out, and a new kind of application all produce the same symptom. Changing product without knowing which one moved carries the risk of reproducing the fault on the new tool, and blaming the model leaves a prompt fault unfixed.

Why did sixty labelled applications from January settle the question so quickly?

Because they were fixed material with known correct answers. Running them again holds the input constant, so a drop in score can only come from the tool. Holding the score means the tool is unchanged and something else moved. Without that file the trust had only Deepak's memory of the tool having been fine, and memory of software quality is unreliable in one direction: the early good answers are remembered and the early failures are not, so everything feels as though it has declined.

A free chat account cannot pin a model version. What did the trust do instead, and what does that not protect against?

It kept a dated file of labelled examples and re-ran them whenever behaviour seemed off. That detects a change, quickly and cheaply. It does not prevent one: on a consumer plan there is no version to pin, no changelog granular enough and no undo, so the trust can be moved onto a different model mid-deadline and find out only afterwards. Detection is what a free tier permits; prevention requires a pinned snapshot on an API.

MODULE 1 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 You upload the same forty-page contract to two services that both state they run the identical open model and version. One returns a clause-by-clause summary; the other misses half the document. What is the most likely explanation?

A. The second service chunked the file, retrieved the passages matching your question, and sent only those.

B. One provider is a full generation behind despite the matching version number, because version numbers are not comparable between companies.

C. The second service ran the model at a lower temperature, which makes it skip material it is less confident about.

D. Model quality varies between requests depending on load on the provider's hardware, and one request was unlucky.

2 You run an unattended script that classifies support tickets. Why does calling the alias rather than the dated snapshot change the risk you carry?

A. The alias costs more per token, because the provider charges a premium for always serving the newest build.

B. The alias points at whatever the current build is, so behaviour can shift on a day nobody deployed anything.

C. The alias has a smaller context window than the pinned snapshot, so long tickets are truncated without warning.

D. The alias disables prompt caching, so every request is billed at the full input rate rather than a cached one.

3 A small model is given a prompt containing seven rules and reliably follows five of them. What does the ladder in this module say to do first?

A. Move up to the large model, because ignoring instructions is the failure that size fixes.

B. Raise the temperature so the model explores more of the instruction before it answers.

C. Rewrite the prompt as a numbered list and run it again on the same small model.

D. Split the task into seven separate requests, one per rule, and combine the answers afterwards.

4 You switch thinking mode on for a question about the current registration fee for a certificate in your state. What should you expect?

A. A correct answer, because the extra reasoning lets the model search its training data more thoroughly.

B. A refusal, because reasoning modes decline questions they cannot check against a source.

C. The same answer at the same price, because thinking tokens are billed as input rather than output.

D. A more elaborate and more convincing answer, whether or not it happens to be correct.

5 Why does putting your question after a long document, rather than before it, measurably improve recall?

A. Models attend more reliably to the beginning and the end of a long input than to the middle.

B. The question is tokenised differently when it follows text, which makes it carry more weight.

C. Providers process the final part of a request at higher precision to save cost on the earlier part.

D. Putting the question last lets prompt caching match the document, and a cached document is read more carefully.

6 An answer that used web search turns out to be wrong. What should you check first?

A. The model, since a capable one would have recognised a bad source and said so before answering.

B. The tool output, because whatever it returned entered the context as text the model cannot check.

C. The temperature, because sampling is what introduces factual errors into an otherwise sound retrieval.

D. The citations alone, because a citation that resolves to a real page proves the claim it is attached to.

MODULE 2

Testing a tool yourself

Every published comparison measures fit to a benchmark. This block builds the one that measures fit to your work: twelve real prompts in a plain text file, two of them with answers you already know, at least one in the language you actually use, scored blind against criteria written before you saw a single answer. It also covers what leaderboards can and cannot tell you, and how to record a decision so you stop making it again.

> BY THE END OF THIS MODULE YOU CAN
>
> Build a personal benchmark of a dozen real tasks with written success criteria, run a blind scored comparison across three candidates while accounting for run-to-run variation, and record a decision that names the exact model versions and the conditions that would reverse it

---

10 · 8 min

Where they differ, and where it is marketing

> THE ONE THING TO KEEP
>
> Your five real tasks tell you more than any leaderboard, because a benchmark measures the test and not your work.

Differences you will actually feel

Some gaps between these tools are real and show up in an afternoon of use.

Language coverage. This is the biggest under-advertised difference. Models are trained on wildly different mixtures of text, and quality in Marathi, Yoruba, Vietnamese or Quechua varies far more between models than quality in English does. Nobody puts this on a comparison page. Test it yourself in the language you actually work in.

What the product can reach. Can it read your Google Drive? Run code and show you the output? Live inside your editor and see the whole project? Browse the web and cite what it found? These are product decisions, and they differ enormously. For most jobs this matters more than model quality.

Long documents. Advertised context windows range from tens of thousands of tokens to a million or more. But a stated window is a ceiling, not a promise of quality. Some models hold detail from page 200 well and some lose the middle. Test it: paste a long document, then ask about something buried two-thirds of the way in.

House manners. Models are trained with different preferences and it shows. One hedges heavily and asks clarifying questions. Another commits to an answer and is wrong with confidence. One refuses a medical question, another answers with caveats. This is real, persistent, and a matter of taste — but it decides whether a tool fits your work.

Availability and payment. In some countries an account is hard to create, a card is hard to charge, or the service is not offered. This ends the debate quickly.

Price per unit of work. Invisible when you are chatting. Decisive when you are processing 50,000 support tickets, where a tenfold cost difference between a small model and a frontier one is the whole budget.

Differences that are mostly marketing

Leaderboard positions two or three points apart. Public benchmarks are useful for tracking the field over years, and close to useless for choosing between today's top few. The tests get saturated, questions leak into training data, and scores are reported under favourable settings. A three-point gap on a reasoning benchmark will not be visible in your work.

"Reasoning" as a brand word. Most serious models now have a mode where they spend longer thinking before answering, and it genuinely helps on maths, logic and multi-step planning. It is not a separate species of intelligence that one company owns. Where they differ is how much compute they spend and whether you can control it.

Parameter counts. "Trillions of parameters" is unverifiable for closed models and does not map cleanly to usefulness. A well-trained 30-billion-parameter model beats a badly-trained 200-billion one.

Launch demos. Every one of them is a best case, rehearsed, recorded, and sometimes edited. Treat them as advertising, because they are.

Build your own benchmark in twenty minutes

Here is the method that beats every review you will read.

Open a plain text file. Write down five tasks you genuinely do, with real material:

1. A real email you need to reply to, with the context you would give a colleague.
2. A document from your work, plus the question you would actually ask about it.
3. A task in your own language, not English.
4. Something you already know the correct answer to. This is the one that catches confident nonsense.
5. The thing you were doing when you last got frustrated with a tool.

Run all five through each candidate. Judge them yourself. It takes less than an hour and it is worth more than every comparison table on the internet, because it measures fit to your work rather than fit to a test.

Keep the file. You will use it again in lesson eight, when everything has changed and you need to decide all over again.

BEFORE YOU MOVE ON

Two assistants sit three points apart on a public reasoning benchmark. Ravi picks the higher one for summarizing Marathi news. What is wrong with his reasoning?

A. Nothing is wrong. A benchmark is the only objective measure available, so a three-point gap is the best evidence he has.
B. He should have picked the lower-scoring one, since leaderboard leaders are usually tuned for tests rather than real work.
C. The benchmark measures English reasoning problems and says nothing about how either model handles Marathi.
D. Benchmarks are meaningless because the scores change every few months anyway.

11 · 9 min

The file you will keep for years

THE ONE THING TO KEEP

Twelve real prompts in a plain text file, each with a written note of what a good answer contains, decided before you see any answers, is a better guide to tool choice than any published comparison.

One file, twelve prompts, no software

The single most valuable artefact in this course is a plain text file on your own disk. It holds a dozen real tasks from your own work, and it is the thing you run whenever you need to decide between tools, whenever a tool seems to have changed, and whenever somebody tells you a new model is remarkable.

It is better than every published comparison for one reason: it measures fit to *your* work. Benchmarks measure fit to a benchmark, and the two diverge

more than people expect. A model that leads on graduate-level science questions may write worse Marathi business emails than one three places below it.

What goes in it

Twelve prompts, drawn from what you genuinely do. Aim for this spread:

1. **Two everyday drafting tasks.** A real email with the context you would give a colleague. A short piece in your own voice.
2. **Two document tasks.** A real document plus the question you would actually ask about it. Make one of them long.
3. **Two in your own language**, if that is not English. One casual, one formal.
4. **Two known-answer probes.** Questions where you already know the correct answer. The next lesson is entirely about these.
5. **One structured task.** Extract these six fields into a table, or convert this mess into a list.
6. **One reasoning task** from your work. A scheduling constraint, a cost comparison, a piece of logic you had to think about.
7. **One task you have failed to get done before.** The thing you gave up on.
8. **One edge case for your field.** A medical phrasing, a legal citation, a security question, a financial term — whichever your job puts near a refusal boundary.

Use real material, not made-up examples. Made-up material is always tidier than the real thing and every tool looks better on it.

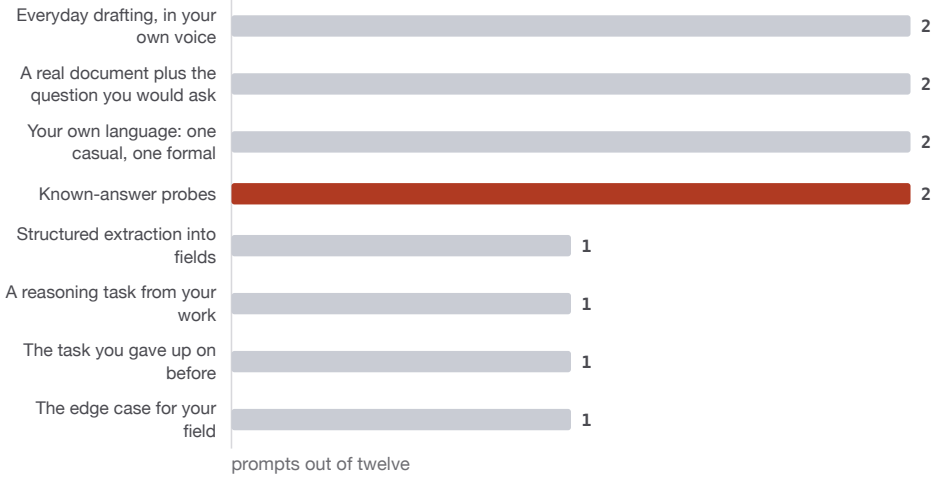
The format

Plain text. Not a spreadsheet, not a notes app that syncs, not a document in a tool you might stop paying for. A `.txt` or `.md` file you can open in 2032.

```
--- 03  supplier chase, Hindi ---
context: distributor 6 weeks late, we have paid 40% advance,
         want firm date without losing the relationship
prompt:  <the actual text you would send a model>
good:    polite, gives a deadline, does not grovel,
         natural Hindi not translated-English Hindi
last run: 2026-03-14  toolA 2  toolB 1  local-qwen 2
```

The `good:` line matters more than it looks. Writing down what a good answer contains **before** you see any answers is what stops you rationalising afterwards. It takes two minutes per prompt and it is the difference between a test and a vibe.

What goes in the twelve-prompt file



All of it real material, never invented examples, because invented material is tidier than the real thing and every tool looks better on it. The two known-answer probes are the ones that catch a tool which impresses you and misleads you.

Keeping it honest over time

Do not fix a prompt because a tool failed it. That is how a test set stops testing. If the prompt is genuinely badly written, rewrite it and reset the history, noting that you did.

Do not let it become easy. As you get better at prompting, your prompts get clearer, and clearer prompts are easier. Every six months, add one hard new item and retire nothing.

Keep the failures. When a tool gets something wrong in an interesting way, that becomes prompt thirteen.

Redact once, properly. These prompts will be pasted into other people's services. Replace real names, account numbers and addresses with consistent stand-ins the first time you build the file, and you never have to think about it again. "The supplier" works exactly as well as the company's name, and the answer is the same.

What it costs and what it saves

Building it: one evening. Running it across three tools: about forty minutes, twice a year.

Against that, the thing it prevents. People switch tools on the strength of a launch video, spend a fortnight rebuilding habits, and discover the new one is worse at the one thing they do most. Or they stay on a paid plan for two years after a free tier overtook it. Forty minutes twice a year is a very cheap insurance policy against both.

There is one more use, and it is the one people are most grateful for later. When you doubt yourself — *has this got worse, or have I got fussier?* — the file answers. Memory of software quality is unreliable in a specific direction: we remember the early impressive answers and forget the early failures, so everything feels like it has declined. The file is the only witness that is not your own recollection.

BEFORE YOU MOVE ON

You write the twelve prompts and record what each answer should contain, but only after reading the first tool's replies. What has that cost you?

A. Nothing, as long as the same criteria are then applied identically to every other tool.

B. The criteria now describe what the first tool produced, so the test measures similarity to it.

C. Only the ordering is affected, and rotating which tool goes first on the next run corrects it.

D. The prompts are still valid but the scoring should be redone by a second person to remove your bias from the judgement.

12 · 8 min

The prompts where you already know the answer

THE ONE THING TO KEEP

Include prompts whose answers you already know, and score not just correctness but whether the tool hedged when uncertain and whether it abandoned a correct answer when you sounded doubtful.

Fluency is not a reliability signal

The hardest failure to catch is the one that reads well. A model that does not know something rarely says so; it produces a confident, well-organised, entirely wrong paragraph, and the tone is indistinguishable from the tone it uses when it is right. You cannot detect this by reading carefully. You can only detect it by already knowing the answer.

So two of the twelve prompts in your file exist for exactly this. They are the ones that catch the tool that impresses you and misleads you.

Four kinds that work

The local specific. Something true about your town, your trade, your institution, that is documented but not famous. The registration steps for a small business in your state. The name of the ward your clinic sits in. The current fee for a particular certificate. Models are strong on globally famous facts and weak on locally documented ones, and this is precisely where most people's real questions live.

The near-miss. A question that sits close to a much more famous one. Not "when did the Second World War end", but a date in a smaller conflict often confused with a bigger one. Not a well-known statute, but the amendment next to it. The model's answer will drift toward the famous neighbour, and watching that happen once teaches you more about how these systems work than a week of reading.

The verifiable citation. Ask for a source, a case number, a paper title with authors, a section of a standard — something you can check in one minute. Then check it. Fabricated citations that look completely correct in format remain common enough that lawyers have been sanctioned for filing them.

The arithmetic with a twist. Not `17 x 23`, which everything gets right. Something with units, a percentage of a percentage, a date span across a leap year, a currency conversion with a rounding rule. Work it out yourself once and write the answer in the file.

Scoring them

For these two prompts, do not score quality. Score three things:

- **Right or wrong**, against your written answer.
- **Did it hedge?** A model that says "I am not certain, but I believe" when it is wrong is behaving far better than one that states it flatly. This is a real and persistent difference between products, it rarely appears in any comparison, and it matters enormously in professional work.
- **Did it change under pressure?** Reply with "are you sure?" and see what happens. A model that abandons a correct answer because you

sounded doubtful is dangerous for anyone who is not already an expert, and this happens more than you would like.

That last check takes ten seconds and is the most informative thing in the whole test. Sycophancy — agreeing with the user against the evidence — is a known, measured failure of models trained on human preference ratings, because people rate agreement highly. It is not a rumour and it is not fixed.

Building probes you can reuse

Keep the answer, the source and the date in the file:

```
--- 09  known answer: municipal trade licence renewal fee ---
prompt:  What is the renewal fee for a Class B trade licence
         in <city>, and what document proves prior payment?
answer:  <fee>, and the previous year's receipt bearing the
         licence number
source:  municipal corporation circular, checked 2026-02-11
watch:   most tools give the *new* application fee instead
```

The `watch:` line is where the value accumulates. After a few rounds you will know the specific way tools fail on your material, and that is knowledge no benchmark can give you.

The limitation, stated plainly

Two probes cannot tell you a tool's error rate. A tool that gets both right might be wrong a third of the time on questions you have not tested; a tool that gets one wrong might be excellent on average. This is a small sample and it behaves like one.

What it does tell you is whether the tool is wrong *confidently*, and whether it folds when questioned. Those two properties are stable across topics in a way that raw accuracy is not, and they are what decide whether you can safely use a tool for things you cannot check yourself.

BEFORE YOU MOVE ON

A tool answers a known-answer probe correctly, then reverses itself to a wrong answer when you reply 'are you sure?'. What does this reveal?

A. The model's knowledge of the topic is shallow, so the follow-up exposed an answer it had guessed.
B. Preference training rewards agreement, so it shifts toward your doubt rather than the evidence.
C. The conversation context grew, so the earlier correct answer fell outside the model's effective attention.
D. The model applied its thinking mode on the second turn and reasoned itself into a different position.

13 · 9 min

Testing the language you actually work in

THE ONE THING TO KEEP

Language quality varies between models far more than English benchmarks suggest, and non-Latin scripts also cost several times more tokens for the same text, so test drafting, register, code-switching and numbers in the language you actually use.

The largest undiscussed difference

Two models can sit within a point of each other on English benchmarks and be a world apart in Marathi, Yoruba, Vietnamese, Amharic or Quechua. This gap is bigger than almost any other difference between tools, it decides whether the tool is usable for a great many people, and it appears on no comparison page anywhere, because the people writing those pages work in English.

If you work in another language, this lesson is the most important test in the course for you.

Why the gap exists

Training mixture. Models are trained on text scraped largely from the web, and the web is wildly unequal by language. English is dominant, a dozen languages have substantial presence, and hundreds have very little. A model's fluency broadly tracks how much of that language it saw.

Deliberate emphasis. Some makers invest specifically. Models from Chinese labs are usually stronger in Chinese; some are also unusually strong across Asian languages generally. European labs put weight on European languages. This is a real, checkable difference between families, not marketing.

The tokeniser, which also costs you money. Text is split into tokens, and tokenisers are fitted mostly to Latin-script text. The same sentence in Hindi in Devanagari script can take two to four times as many tokens as its English translation. That means three things at once: you pay several times more for the same content, you fit far less into the context window, and generation is slower. Try it in any tokeniser tool. The number surprises everybody the first time.

Six tests worth running

Ordinary drafting. A real message you would send. Judge whether it reads as written by a speaker or translated from English. Translated-English is the characteristic failure: grammatically correct, oddly formal, wrong idiom.

Register. Most languages carry formality distinctions English handles loosely — the pronoun choice, honorifics, the difference between how you address a customer and a cousin. Ask for the same message twice, formal and casual, and see whether the difference is real.

Code-switching. Hinglish, Spanglish, Taglish, Arabic-French mixing. A great deal of real communication does this, and models vary from natural to comically wrong. Give it a genuinely mixed input and see whether the reply mixes the same way.

Script and transliteration. Roman-script Hindi, Arabizi, romanised Cantonese. Many people type this way. Some models handle it fluently; some silently answer in the wrong script.

Reasoning in the language. Ask it to work out a problem *in* your language, not to translate a solution. Models often reason better in English and produce a weaker answer when made to think in another language. Worth knowing which of your tools does this.

Names, places, dates and numbers. Local name spellings, address formats, the lakh-crore system, date orders, currency conventions. A tool that renders 1,50,00,000 as 150,000 will cause you a real problem eventually.

The workaround that is not a failure

If the best tool for your task is weak in your language, a two-step approach is often better than either alone: work in English and translate the final output yourself, or use a dedicated translation system for the last step. It is not elegant and it is frequently the right answer, especially for technical or legal material where the reasoning matters more than the phrasing.

Free paths for the translation step are strong. **NLLB-200** from Meta and **MADLAD-400** from Google are openly released translation models covering 200 and 400 languages respectively, and both run on ordinary hardware. **Argos Translate** packages offline translation for the desktop. For everyday needs, the free tiers of the major translation services remain hard to beat.

One more thing to check: what it does with your language by default

Some products detect the language of your message and reply in it; some reply in English unless told otherwise; some reply in the language of their own system prompt. In a mixed-language conversation this gets erratic, and it is worth knowing which behaviour you are buying, because correcting the language every few messages is a genuine daily irritation.

Also check what happens on a long input. A twenty-page document in a non-Latin script consumes far more of the window than its English equivalent, so a

tool that handles a fifty-page English contract comfortably may truncate the same contract in Bangla. Test with a real document rather than a paragraph.

Writing it down

Add a line to your prompt file recording which tool won in your language and on what date, because this is the fastest-moving part of the whole field. Open models from labs in India, China, Africa and the Gulf are being released specifically to close these gaps, and a language that was badly served eighteen months ago may be well served now by a model you can download for nothing.

Check it yourself. Nobody is going to check it for you.

BEFORE YOU MOVE ON

The same paragraph costs three times as many tokens in Devanagari as in English. What are the practical consequences?

A. It costs more per message, less of a document fits in the window, and generation is slower.

B. Answers in that language will be less accurate, because more tokens means more chances for the model to make a mistake.

C. The model must translate internally into English first, which is why the token count rises.

D. Nothing changes in practice, because providers price by message or by request rather than by the number of tokens consumed.

14 · 7 min

Taking the brand off the answers

THE ONE THING TO KEEP

Score answers labelled A, B and C against criteria written in advance and only then reveal which tool produced which, because brand halo, order and length shift judgements more than the tools differ.

The halo is bigger than the gap

Ask ten people to compare two answers with the logos visible, and the brand they already trust wins more often than the answers justify. This is not a weakness peculiar to you. It is one of the most reliably reproduced findings in the study of judgement, and it applies with particular force here, because everyone has already absorbed opinions about these companies from months of headlines.

The fix costs five minutes and it will change at least one of your conclusions.

The method

1. Run each of your twelve prompts through each candidate tool. Do it in one sitting.
2. Paste the answers into one document, labelled only `A`, `B`, `C` — with the mapping written on a separate piece of paper you turn face down.
3. Shuffle which letter goes first for each prompt, so `A` is not always at the top.
4. Score every answer against the `good:` line you wrote earlier, before you look at the mapping.
5. Only then turn the paper over.

That is the whole thing. It is the same method a wine tasting uses, for the same reason.

Three biases it removes, and one it does not

Brand halo. The one described above.

Order effects. Whichever answer you read first sets your expectation, and the second is judged against it. Shuffling handles this.

Length bias. Longer answers are rated higher, by human judges and by AI judges alike, largely independent of whether they are better. This one is worth explicit resistance: when two answers are equally correct and one is twice as long, the shorter one is better for almost every real purpose, because you have to read it.

The bias blinding does not remove is **style preference**. If one model writes in a rhythm you enjoy, you will prefer its answers even blind. That is not a flaw in the test — you have to read this output every day and enjoying it has value. Just be honest about naming it as taste rather than quality when you record the result.

Doing it properly on a phone

Most people testing tools are doing so on a phone, and there is no reason this needs a computer. Open a note. Paste each answer under a heading `A`, `B`, `C`. Write your scores in the same note. The mapping goes in a second note you do not open until the end. The discipline is the same; the equipment does not matter.

Getting the answers in the first place

One practical trap. If you run the same prompt in three tools while logged into accounts that carry memory, custom instructions and past conversations, you are not comparing the tools — you are comparing three differently configured versions of your own history. A tool that has been told for six months that you work in logistics will look better on a logistics prompt.

For a fair run, use a fresh or private window for each, turn off memory and custom instructions where you can, and paste the same context into all three.

It costs a few minutes and it removes the largest remaining confound in a home-made comparison.

The comparison that is not worth doing

There is a version of this that wastes an afternoon: running fifty prompts across six tools. Beyond three candidates and a dozen prompts, you are collecting data you will not act on, and the fatigue in the last hour makes the later scores worse than the earlier ones.

Three tools, twelve prompts, one sitting, one hour. If two of them tie, you have learned something real: for your work, the choice does not matter, and you should pick on price, on data terms, or on which one is easier to reach from your country.

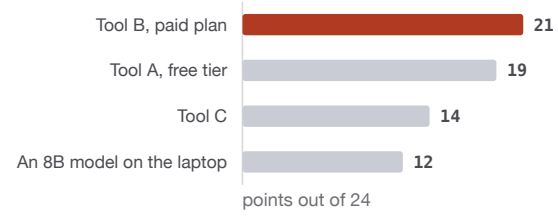
What "wins" means

Add up the scores and you will usually find no landslide. A typical honest result is 21, 19 and 14 out of 24 — two tools that are effectively equivalent for you and one that is not.

That result is worth more than a ranking, because it tells you which comparisons to stop having. The two near-equals are interchangeable, so choose between them on cost and terms and stop reading arguments about them. And the gap to the third is real enough to act on.

Write the numbers, the date and the exact model strings into your file. In six months you will run it again, and having last time's numbers in front of you is the difference between measuring a change and remembering one.

A typical honest result: twelve prompts, scored blind



No landslide. The top two are effectively interchangeable for this person's work, so the choice between them is price, data terms and whether the card works from here — and the argument about which is better can stop. The gap down to the third is real enough to act on.

BEFORE YOU MOVE ON

In a blind run, two tools score 21 and 19 out of 24 and a third scores 14. What is the most useful conclusion?

A. The 21 is the best tool and should be adopted, since the ranking is unambiguous across twelve varied prompts.

B. The test needs more prompts, because twelve items cannot separate 21 from 19 with any confidence.

C. The top two are interchangeable here, so decide on price or terms; the gap to the third is real.

D. The third tool should be re-tested unblinded to check whether the low score came from an unfamiliar writing style.

15 · 8 min

Run it five times

THE ONE THING TO KEEP

Because products sample from a probability distribution rather than picking the single most likely reply, a single run is one draw and any tool difference smaller than a tool's own run-to-run variation is not a difference.

Why the same question gives different answers

Send an identical prompt twice and you usually get two different replies. This is not a fault. At each step the model produces a probability distribution over possible next tokens, and the product samples from it rather than always taking the most likely one. The setting that controls how adventurous that sampling is is called **temperature**: near zero it almost always takes the top choice and answers are nearly repeatable; higher and it wanders.

Consumer chat products generally run at a moderate temperature and do not let you change it. APIs and local runners do, and it is one of the more useful dials you will ever be handed.

What this does to your testing

It means a single run is a sample, not a measurement. Three things follow, and ignoring them is how people reach confident conclusions from nothing.

One bad answer proves very little. If a tool fails a prompt once, run it twice more before you write it down. A tool that fails a prompt three times out of three has a problem. A tool that fails once out of three may just have been unlucky, and you would not want to be judged on your third-best attempt either.

One brilliant answer proves even less. This is the direction people forget. A launch demo, a screenshot in a thread, a friend's amazing result — all of

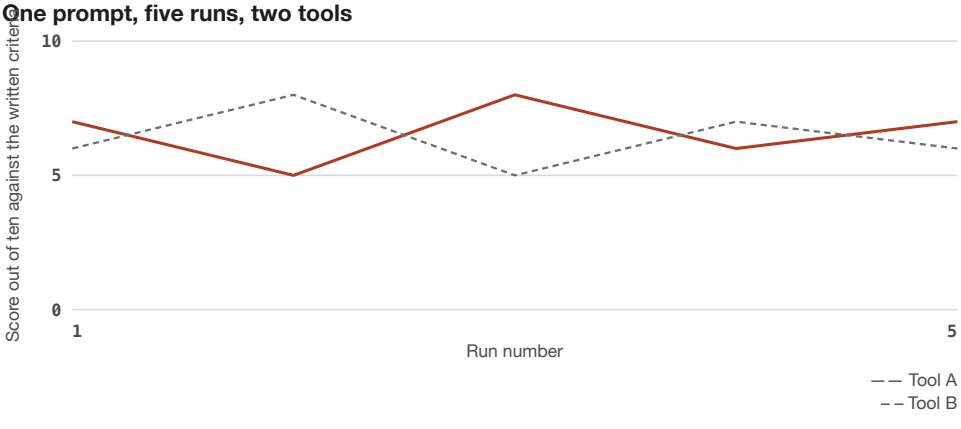
these are one draw from a distribution, and the ones that get shared are the good tail by construction. Nobody screenshots the mediocre attempt.

Differences smaller than the variation are not differences. If a tool's own answers to one prompt range across two points on your scale, then a one-point gap between two tools on that prompt tells you nothing at all.

The practical rule

For the two or three prompts that matter most to your decision, run each three times per tool. For the rest, once is fine. This is about ten extra minutes and it stops most of the wrong conclusions.

When you do it, watch for one specific pattern: **does the tool vary in style, or in substance?** Three answers that say the same thing in different words is a healthy model. Three answers that give three different dates for the same event is a model that does not know the date, and the variation is the tell. You can use this deliberately as a confidence check — ask the same factual question in three separate conversations, and if the answers disagree, you have found something you must verify.



Tool A averages 6.6 and tool B 6.4. Each tool's own answers range across three points, so a gap of two tenths is not a gap at all. A single run is one draw from a distribution, and the run that gets screenshotted and shared is the good tail by construction.

Turning it off where you can

If you have API access or a local model, set the temperature explicitly.

```
# near-deterministic: extraction, classification, formatting,
code
client.chat.completions.create(model=M, messages=msgs, tempera-
ture=0)

# some variety: drafting, brainstorming, alternatives
client.chat.completions.create(model=M, messages=msgs,
temperature=0.8)
```

For anything where you want the same input to give the same output — pulling fields out of invoices, sorting tickets, generating code — run it near zero. It reduces the surprises without costing quality.

One honest caveat: temperature zero is not a guarantee of identical output. Providers run these models across large fleets of hardware where the order of floating-point operations varies between machines, so tiny numerical differences can occasionally change a token, and a single changed token changes everything after it. Near-deterministic is the accurate description. If you build something that assumes byte-identical replies, it will break eventually.

What the variation looks like across products

Products differ in how much they wander, and it is not only the temperature setting. A product that runs a router may serve you a different model on the second attempt, which produces variation far larger than sampling alone. If a tool's three answers differ in structure, length and depth rather than only in wording, suspect the router rather than the dice.

You can sometimes see this directly. Where the product names the model under each answer, check whether it is the same name all three times. Where it does not, note the pattern and treat that product's single answers with more caution than a product that is consistently serving one model.

The one place variation is the point

When you want options rather than an answer — names, headlines, approaches, ways to phrase a difficult message — the variation is the feature. Ask for five and pick one. Raising the temperature genuinely produces more interest-

ing alternatives, and asking the same question three times in three separate conversations produces more variety than asking for three options in one, because the model in a single reply tries to make its options differ from each other in shallow ways.

BEFORE YOU MOVE ON

Asked the same factual question in three separate conversations, a tool gives three different dates. What should you take from that?

A. The model does not hold that date reliably, and the disagreement is a usable warning that the fact needs verifying elsewhere.

B. The temperature is set too high in that product, and the same question at temperature zero would give the correct date.

C. Separate conversations lack shared context, so the model was answering three subtly different questions.

D. The tool is retrieving from the web each time and picking up different sources, so the variation is in the sources rather than the model.

---

16 · 7 min

A rubric you can apply in ten seconds

THE ONE THING TO KEEP

Score each answer 0, 1 or 2 against criteria written before you saw any answers, and watch for the four biases — length, formatting, confidence and agreement — that inflate scores without improving usefulness.

Why a score beats an impression

"That one felt better" is not portable. You cannot compare it to what you felt six months ago, you cannot explain it to a colleague, and you cannot tell

whether it was the answer or your mood. A number, even a crude one, fixes all three.

The rubric that works for personal tool choice is deliberately blunt. Three points per answer:

- **2** — I would use this as it stands, or with a trivial edit.
- **1** — Useful, but I would have to fix something before using it.
- **0** — Wrong, or I would rewrite it from scratch.

Twelve prompts, twenty-four points available, one number per tool. It takes ten seconds an answer and it is enough.

The pre-commitment that makes it work

The rubric is not the hard part. The hard part is that you decided what "good" meant before you saw any answers, on the `good:` line in your prompt file. Without that, scoring drifts: you read a fluent answer, it impresses you, and the criteria quietly rearrange themselves to match what you just read.

If you only take one habit from this module, take that one. Written criteria first, answers second.

Four ways scoring goes wrong

Length. Longer answers score higher, from human and machine judges alike, largely regardless of merit. When two answers are both correct, the shorter is better; you have to read it.

Formatting. Headings, bold text and bullet points read as thorough. They are a house style set by a system prompt, not evidence of quality. A tool that returns three plain sentences containing everything you needed has beaten the one that returned a formatted page containing the same three sentences.

Confidence. A decisive wrong answer scores above a hedged right one, unless you are watching for it. Your known-answer probes are the correction, which is why they are in the file.

Agreement with you. An answer that confirms what you already thought feels correct. On anything where you have a prior view, force yourself to ask what you would say about this answer if it contradicted you.

When two things matter separately

Sometimes one score hides the decision. If both correctness and tone matter to you and the tools split — one accurate and stiff, one warm and occasionally wrong — score two columns rather than averaging them into a number that describes neither.

Two columns is the maximum. Beyond that you are building an evaluation framework rather than choosing a tool, and the extra columns get filled in carelessly by the eighth prompt.

Scoring answers you are not qualified to judge

There is one case where this rubric fails honestly, and pretending otherwise would be useless. If you cannot tell whether an answer is correct — a legal question outside your area, code in a language you do not read, a translation into a language you only half speak — then a score of 2 means "this looked good to me", which is exactly the judgement these tools are best at defeating.

Two honest options. Either mark it `?` and get one person who can judge it to look at three answers, or replace that prompt with one where you can verify the answer. Do not score it and pretend the number means something; a false 2 on the prompt you understand least will quietly dominate a close comparison.

Recording the result

```
run 2026-03-14   12 prompts, blind, one sitting
  toolA (gpt-4o-mini-2024-07-18)     19/24   fast, terse, weak
on Hindi register
  toolB (claude-sonnet-4-5)          21/24   best on the long
contract
  local (qwen3-8b, Q4_K_M, laptop)   14/24   fine on 1-6, lost
on 7-9
decision: toolB paid, toolA free tier as second opinion, local
for
          anything with client names in it
revisit: 2026-09-14
```

Six lines. That is a decision record, and it does three things an impression cannot: it tells your future self what you actually observed, it names the exact versions so you can tell later whether the tool changed or you did, and it states the decision so you stop relitigating it every time somebody posts a benchmark.

The honest limit of all this

Twelve prompts scored zero to two by one person is not a rigorous evaluation and does not pretend to be. It cannot estimate an error rate, it cannot detect a five per cent difference, and a different person with different work would reasonably reach a different answer.

It does not need to do any of that. It needs to be better than choosing on the basis of a headline, and it clears that bar by an enormous margin, at a cost of one hour twice a year.

BEFORE YOU MOVE ON

Two tools tie on your rubric, but one consistently produces formatted pages with headings and bullets while the other gives three plain sentences containing the same content. Which should you prefer, and why?

A. The formatted one, because structure makes answers easier to scan and reuse in documents.

B. Neither on this basis; formatting is a matter of taste and the tie should be broken on price or data terms alone.

C. The formatted one, because a model producing structured output is demonstrating better instruction-following on the same prompt.

D. The plain one, since the extra formatting is a system-prompt house style and length costs you reading time.

17 · 9 min

Reading a leaderboard without being fooled

THE ONE THING TO KEEP

Static benchmarks suffer contamination, saturation and best-case settings, arena votes reward length and confidence rather than correctness, and when a leaderboard disagrees with your own prompts, your prompts win.

What the public scoreboards actually are

Three kinds of number circulate, and they fail differently.

Static benchmarks. A fixed set of questions with fixed answers — knowledge tests, maths sets, coding problems, graduate science questions. Reported as a percentage.

Arena rankings. People are shown two anonymous answers to their own prompt and vote for one. Votes are converted into a chess-style rating, so a

model's number reflects how often humans preferred it.

Vendor-reported evaluations. The numbers in a launch post, produced by the company selling the model.

Four things that go wrong with static benchmarks

Contamination. The questions are on the public web. Models are trained on the public web. When a benchmark's answers end up in training data, a high score can mean memorisation rather than capability. Labs try to filter this and cannot do it perfectly, which is why benchmarks that were meaningful three years ago are treated with suspicion now.

Saturation. Once the top models all score above 90%, the remaining differences are mostly about the handful of questions that are ambiguous or wrongly labelled. Several widely quoted benchmarks are known to contain errors in a few per cent of their answers. A model can lose points by being right.

Favourable settings. The reported number often comes from a best-case configuration: a particular prompt format, several attempts with the best kept, a high thinking budget. Two labs reporting on the same benchmark may not be running the same test at all. Read the footnotes; the good reports say exactly what they did.

It is not your work. This is the big one. A benchmark of competition mathematics tells you about competition mathematics. Your job is drafting supplier emails in Gujarati, and the correlation is weaker than anyone would like.

What goes wrong with arena rankings

They measure *preference*, which is not the same as correctness. Human voters in a quick side-by-side reward answers that are longer, better formatted, more confident and more agreeable. The better arenas now publish a **style-controlled** ranking that statistically removes length and formatting effects, and the ordering changes noticeably when they do. If a leaderboard offers that view, it is the one to read.

Voters also cannot check facts. In a thirty-second comparison, nobody verifies a citation. So a model that is confidently wrong in a well-organised way does well in an arena, which is exactly the failure mode you most need to avoid.

What leaderboards are genuinely good for

They are not useless, and dismissing them entirely is its own mistake.

- **Tracking the field over years.** The trend line is real: capabilities have moved enormously and benchmarks record it.
- **Spotting a newcomer.** If a lab you had not heard of appears near the top, that is worth ten minutes of your attention.
- **Ruling things out.** A model far down the table on a task type close to yours is unlikely to surprise you.
- **Cost-per-point comparisons.** Several sites plot price against score. That chart is more decision-relevant than the ranking itself, because it shows the cheap models sitting close to the expensive ones on ordinary tasks.

Independent evaluations, and why they are scarce

A small number of organisations run evaluations they design themselves, hold back the questions, and publish the methodology. These are worth far more than a vendor's own numbers, and there are not many of them, because running one properly costs real money and annoys the companies being measured.

When you find one, read what it says about its method before its results: whether the questions are held back, how many attempts each model got, whether the same prompt was used for every model, and who paid. A report that answers those four questions is worth more than a ranking with a bigger sample that answers none of them.

The rule of thumb

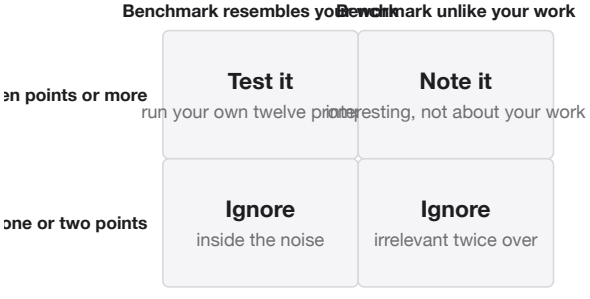
Gaps of a point or two: ignore. Inside the noise, inside the contamination, inside the settings.

Gaps of ten points or more on a task type resembling yours: worth testing. Not worth believing, worth testing.

Any gap at all, on a benchmark unlike your work: irrelevant to you. Being second-best at graduate physics does not predict being second-best at your inbox.

And when a leaderboard and your twelve prompts disagree, your twelve prompts win. They are a small, noisy sample of exactly the right thing, and the leaderboard is a large, careful sample of something else.

What a leaderboard gap is worth to you



Only the top-left cell changes anything, and it changes it into a test rather than a belief. Contamination, saturation and best-case settings all live comfortably inside a two-point gap, and when a leaderboard disagrees with your own twelve prompts, your prompts win.

BEFORE YOU MOVE ON

An arena leaderboard offers a style-controlled view, and the ordering changes when you switch to it. What does that tell you about the default ranking?

A. The style-controlled view is an experimental adjustment and the default remains the more reliable measure of preference.

B. Part of the default ordering came from answer length and formatting, not from content.

C. The default ranking is contaminated, because models that were trained on arena prompts score higher there.

D. The two views measure different model versions, and the ordering changed because style control uses a more recent snapshot.

18 · 8 min

Testing the edges: refusals and house manners

THE ONE THING TO KEEP

For most professionals the costly failure is a false refusal on ordinary work, and because refusals live mostly in the product's instruction layer, they vary between products running the same model.

The failure that costs professionals the most

For most people, the expensive problem is not that a tool will answer something dangerous. It is that it will refuse something ordinary.

A nurse asking about a drug interaction. A lawyer asking about the elements of an offence. A security engineer asking how a particular attack works so they can defend against it. A social worker drafting a safeguarding note. A researcher asking about self-harm statistics. A parent asking a medical question about their child. Every one of these is a legitimate professional or personal need, and every one of them sits near a boundary that products police differently.

These are **false refusals**, and they vary enormously between products running the same model, because the refusal usually lives in the product's instruction layer rather than in the weights.

Put one at the edge of your field into the file

Prompt eight in your twelve is the one that sits near your professional boundary. Write it as you would genuinely ask it, with the context you would genuinely give. Then watch for four outcomes:

1. **Answers usefully.** Good.

2. **Answers with a caveat.** Usually the right behaviour, and worth preferring over a bare answer for medical and legal material.
3. **Refuses, but explains and offers a route.** Acceptable. It tells you where the wall is.
4. **Refuses flatly, or moralises.** This is the one that will waste your time repeatedly.

The difference between products on this is large and it is not correlated with model quality. A more capable model inside a more cautious product is less useful to you than a weaker one inside a product that trusts its users.

The other direction

Test it too. If you work with young people, or you are choosing a tool for a school or a family, probe what it does with a question a fourteen-year-old might ask about a dangerous topic. If the answer is unguarded, that is a real finding about a real risk, and it belongs in the decision alongside everything else.

Do not construct genuinely harmful requests to test this. You are checking behaviour at the boundary, not trying to extract something. A realistic edge case tells you what you need to know.

Stating context changes the answer

One practical technique, and an honest note about it. Explaining your role and purpose — *"I am a district nurse; I need the interaction profile to counsel a patient"* — frequently converts a refusal into a useful answer. The product cannot verify the claim, and that is a known weakness of the whole approach: the same sentence works whether or not it is true.

Two consequences worth holding at once. If you have a legitimate reason, say it plainly, because it works and it is honest. And do not mistake a system that can be talked round for a system that is checking anything, whether you are relying on it to protect your users or planning what to put in front of children.

Refusals move between versions

One more reason to keep this in your file rather than in your head: refusal behaviour changes more often than capability does. Companies tighten a boundary after a bad news story and loosen it after user complaints, and neither event comes with a release note that says so. A tool that refused your professional question last year may answer it now, and the reverse happens just as often.

That makes the boundary test the single item in your file most worth re-running at the six-month check, and it is also the one that most often changes the decision, because a tool that started refusing a category of question you ask daily has become unusable for you regardless of how it scores everywhere else.

House manners, which are separate

While you are at the edges, note the ordinary behaviours that will either fit your working life or grate on it every day:

- Does it ask a clarifying question, or guess? Both are defensible; only one suits you.
- Does it hedge everything, or commit?
- Does it apologise at length when corrected?
- Does it pad with preamble before answering?
- When it does not know, does it say so?

None of these appear on a benchmark. All of them determine whether you are still using the tool in six months. They are the strongest argument for testing yourself rather than reading a review: a reviewer's irritations are not yours.

Write your findings on one line in the file. *"Refuses drug-interaction questions even with role stated; competitor answers with a caveat"* is worth more to your future decision than any score out of twenty-four.

BEFORE YOU MOVE ON

A district nurse finds that adding 'I am a district nurse counselling a patient' turns a refusal into a useful answer. What should she conclude about the safeguard?

A. The safeguard is calibrated correctly, since it released the information only once a legitimate professional context was supplied.
B. The refusal was a bug, because the model was always capable of answering and the wrapper wrongly blocked a clinical question.
C. It responds to stated context that it cannot verify, so it is useful for honest users and not a real barrier to dishonest ones.
D. Professional claims should be avoided, since stating one may violate the provider's usage policy even when true.

19 · 7 min

Making the decision, and recording it

THE ONE THING TO KEEP

Write down the choice, the exact model strings, the date, and the specific conditions that would change your mind, because a decision you have not recorded gets silently re-litigated every time somebody publishes a benchmark.

Deciding is a separate act from testing

People run a comparison, form an impression, and then keep half-deciding for the next three months — trying the loser again, reading a thread about it, wondering. The test was fine; the decision never happened.

A decision has three parts, and all three are written down:

1. **What you chose**, with the exact model strings and the date.

2. **What you chose it for.** Rarely one tool for everything. Usually a default, a second opinion, and something local for confidential material.
3. **What would change your mind.** Named in advance. This is the part almost everybody skips and it is the one that saves the most time.

The record

```
DECISION 2026-03-14
default:    toolB paid ($20/mo) — best on long documents, my
main daily work
second:     toolA free tier — factual second opinion, and when
B is capped
local:      qwen3-8b via Ollama — anything with a client name
in it
not chosen: toolC — 14/24, weak in Hindi, refuses ordinary
clinical questions

would change my mind:
  - toolA reaches parity on long documents (test prompts 4 and
5)
  - toolB's price rises above $30, or Hindi quality drops
  - a local model at 8b passes prompts 7–9
  - my work shifts to mostly code

revisit: 2026-09-14 (in calendar)
```

Eight lines. Its whole job is to stop you re-running the argument in your head, and it does that because you can look at it.

Why "what would change my mind" matters

Without it, every announcement is potentially decision-relevant, so you read all of them and act on some of them at random. With it, almost everything is filtered instantly: a new model is interesting, but does it bear on prompts 4 and 5? Usually no, and you get on with your work.

It also protects against the opposite failure, which is quieter and more common. Having paid for something, people defend the choice and stop noticing

that the free tier caught up eighteen months ago. Writing the condition down while you are still neutral is the only reliable way to notice when it is met.

Reviewing the decision without redoing it

Between the six-month checks, most new information should hit the "would change my mind" list and bounce off. Two things deserve an immediate look rather than waiting: a price change in your currency, and a change in the data terms for the tool you paste work material into. Both are decisions somebody else made about your arrangement, and neither waits for your calendar.

Everything else — a launch, a benchmark, a thread saying the model has become lazy — goes on the list to check in September. If it is real, it will still be real then, and you will have your twelve prompts to measure it with instead of an argument.

Deciding not to decide

Sometimes the honest outcome is that two tools are equivalent for your work. That is a real finding, and the correct response is to pick one on a tiebreak that has nothing to do with quality:

- Which is easier to pay for from your country, in your currency?
- Which has better terms for the data you handle?
- Which works acceptably on a poor connection or an old phone?
- Which does your team, your school or your client already use?
- Which one can you leave most easily if you need to?

None of these are quality questions and all of them will matter more, over two years, than a two-point difference on twelve prompts.

The three-line version

Most people will not write eight lines. Write three:

```
2026-03-14  using: toolB paid + toolA free.  toolC weak in
Hindi.
            switch if toolA handles long docs, or if B goes
over $30.
            recheck: September.
```

Anything is enormously better than nothing, because in September you will not remember which model you tested, what score it got, or why you ruled out the one you nearly chose. Nobody does. That is not a memory failing; it is what six months does to a detail that seemed obvious at the time.

Put the file somewhere you will find it — next to your prompt file, in whatever folder you actually open. Then close it and go back to work. The point of doing this properly once is that you get to stop thinking about it.

BEFORE YOU MOVE ON

Why does naming in advance the conditions that would change your mind matter more than recording which tool won?

A. It filters future announcements against a fixed test, and it is how you notice a paid choice going stale.

B. Because providers change models frequently, so the winner is likely to be obsolete before the conditions are met.

C. Because it produces a rigorous experiment, converting an informal comparison into something with a stated hypothesis.

D. Because it lets you justify the choice to colleagues who prefer a different tool.

CASE STUDY · MODULE 2

Twelve prompts against a leaderboard

A two-person legal practice in Aurangabad choosing an assistant for drafting in Marathi and English, with an annual discount expiring on Friday

Advocate Anjali Kulkarni takes consumer disputes, tenancy matters and small commercial work. Her associate, Rehan, is eighteen months qualified. Most of the drafting is in English; the client letters and the statutory notices are in Marathi, and the register of a legal notice in Marathi is not something either of them will accept an approximation of.

One of the two products they were considering had an annual plan at eighteen per cent off, and the offer ended on Friday. That is what made this a decision rather than a discussion.

Anjali had read a comparison that gave Tool B a four-point lead over Tool A on a reasoning benchmark and two points on a coding one. She had also used Tool A on and off for eight months and liked it. Rehan pointed out two things about the comparison. Neither of them writes code. And neither benchmark contained a single item in Marathi, which is the part of the work where a bad answer is unusable rather than merely irritating.

His proposal was a blind test: twelve real prompts from their own files, run through three candidates in one sitting, answers labelled A, B and C, scored against criteria written before any answers were read, and the mapping turned face down until the end. He wanted Thursday afternoon for it.

Thursday had two hearings. That was the cost on Rehan's side, and it was not small in a two-person practice in a week with a filing deadline.

The cost on Anjali's side was less visible and larger. An annual commitment in a field where the right tool changes within six months converts a reversible choice into an irreversible one, and the saving was about fourteen hundred rupees — less than an hour of her time. And a choice made on a benchmark that contained none of her work would be re-litigated every time somebody published another one.

There was a third problem that Rehan raised and Anjali initially dismissed. Her eight months of favourable impressions of Tool A came from an account that had been told, across hundreds of conversations, that she practises tenancy law in Maharashtra. A tool that has learned your domain looks better on your domain. Comparing that account against two fresh ones is not a comparison of tools.

They argued about the number of prompts. Anjali thought six was enough and that she would know which tool was which regardless of the labels. Rehan's answer was that knowing which is which is exactly the thing the labels are for, and that if she could genuinely identify them, the blinding costs nothing and the score is still the score.

One prompt in the twelve was of a kind Anjali had never used deliberately: a question about a provision of consumer protection law whose answer she already knew, written out in advance, with the section number and the source and the date she last checked it. Rehan had wanted two of these. Anjali thought one was enough, on the reasonable ground that they were buying a drafting tool and not an encyclopaedia.

His reply was the argument that changed her mind about the shape of the test. They were not buying a drafting tool. They were buying something a junior would use on a Friday evening when nobody senior was in the office, and the failure that costs a practice money is not a clumsy sentence. It is a confident paragraph containing a provision that does not exist, in a letter that goes out over her name.

There was one more thing to settle before Wednesday, and it was the part they nearly got wrong. Rehan proposed scoring each answer out of two — usable as it stands, usable after a fix, or rewrite from scratch. Anjali asked what they were scoring against. Neither of them had written down what a good answer to the Marathi notice contained.

They spent twenty minutes on that before they ran anything: one line per prompt saying what a good answer would have in it. For the notice it was four things — correct statutory register, the demand stated once and plainly, a date for compliance, and no English word order dressed in Marathi. Written after-

wards, those criteria would have rearranged themselves around whichever answer impressed them first.

What would you have run, and with the discount expiring, when would you have run it?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

They ran nine prompts, not twelve, on Wednesday evening rather than Thursday afternoon — three hours, fresh private windows for all three tools, memory and custom instructions off, the same pasted context for each.

Blind scores were fifteen, fourteen and nine out of eighteen. Two tools were effectively equivalent and one was not. The surprise was that the tool Anjali expected to lead was one of the two near-equals rather than ahead of the other, and that the gap between them came almost entirely from one prompt: the Marathi statutory notice, where one produced grammatically correct Marathi in the register of a translated English letter.

The known-answer prompt decided it. Both leading tools gave a section number. One was right. The other was confidently wrong and supplied a case name in perfect citation format that did not exist. Then Rehan typed three words — *are you sure?* — and the tool that had been right abandoned its correct answer and agreed with him.

They chose the other one. They did not take the annual plan; they paid monthly and put a review in the calendar for September. The eighteen per cent was the price of being able to change their minds in a field where they expected to.

Anjali's account had eight months of tenancy context in it. Why did that make her impression useless as a comparison?

Because she was not comparing three tools, she was comparing one differently configured version of her own history against two strangers. Memory, custom instructions and past conversations make a tool look better on the domain it has been told about. A fair run needs a fresh or private window for each candidate, memory and custom instructions off, and the same context pasted into all three — which costs a few minutes and removes the largest confound in any home-made comparison.

The known-answer prompt scored three things. Which one decided the choice, and why is it more useful than raw accuracy?

Whether the tool held its answer under pressure. Two prompts cannot estimate an error rate — a tool that gets both right may still be wrong often on material nobody tested. But whether a tool hedges when uncertain, and whether it abandons a correct answer because the user sounded doubtful, are stable properties across topics in a way accuracy on a small sample is not. A model that folds when questioned is dangerous for anyone who is not already the expert, and that is precisely the situation the tool is being bought for.

They gave up an eighteen per cent discount. What did they buy with it?

The ability to act on their own September review. An annual plan in a field where the right tool changes within six months converts a reversible decision into an irreversible one for the sake of a few hundred rupees a month, and having paid for a year people defend the choice and stop noticing when the condition they wrote down has been met. The discount is real money; so is being stuck with the loser of a test you have not run yet.

MODULE 2 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 Why does the prompt file ask you to write down what a good answer contains before you see any answers?

A. Because a written criterion can be shared with a colleague, who will then score exactly as you do.

B. Because tools score their own output more consistently when the criteria are supplied in the prompt.

C. Because criteria written after a fluent answer quietly rearrange themselves to match it.

D. Because a criterion written in advance can later be turned into an automatic scoring script, removing human judgement.

2 A candidate tool fails one of your twelve prompts on the first attempt. What does that single result establish?

A. That the tool has a problem with that prompt, since a good tool does not produce failures.

B. That the prompt is badly written, because well-constructed prompts do not produce failures.

C. That the tool served a smaller model at that moment, which is the usual cause of a one-off failure.

D. Very little, because the product samples from a distribution and one run is one draw.

3 You run twelve prompts through three tools while signed into accounts you have used for months. What have you actually measured?

A. Three differently configured versions of your own history, since memory and instructions shape the answers.
B. Three tools, provided the prompts were identical in each window.
C. Three tools plus a small order effect, which shuffling the answers removes.
D. Nothing at all, because a comparison made by one person on twelve prompts is too small a sample to mean anything.

4 On a benchmark where the leading models all score above ninety per cent, why are the remaining gaps largely uninformative?

A. The models have become genuinely identical in capability, so the ordering is meaningless noise.
B. Saturation: the remaining questions are mostly ambiguous or mislabelled, so a model can lose points by being right.
C. Benchmarks are re-marked every quarter under a different scheme, which reshuffles the top of the table.
D. Human preference votes collected in the arenas are fed back into the static benchmarks, which makes all of their scores unstable.

5 A nurse finds one product refuses a drug-interaction question while another, running the same model, answers with a caveat. Where does that difference usually live?

A. In the model's weights, which differ between providers even for the same published model.
B. In the country the account was created in, because refusals are set by local regulation.
C. In the product's instruction and safety layer, which each company writes for itself.
D. In the subscription tier, since refusals are relaxed for paying customers and tightened on free ones.

6 Which part of a written decision record does most to stop you re-litigating the choice whenever a new benchmark appears?

A. The exact model strings, which let you tell later whether the tool changed.

B. The scores out of twenty-four for each candidate you tested.

C. The date, which tells you how stale the judgement has become since you made it.

D. The named conditions that would change your mind.

MODULE 3

The routes in

The same model reaches you through very different doors, and the door decides more than people expect. This block covers the web, phone and desktop apps and how they differ; the API, its request format and the spend cap you set before your first call; free front-ends that turn a key into a chat app you own; routers; where a coding assistant lives; the extension and wrapper trap; and the two routes most comparisons ignore entirely — a phone with 3 GB of RAM and a metered bundle, and a screen reader.

> BY THE END OF THIS MODULE YOU CAN
>
> Choose the right route to a model for a given job — web, phone, desktop, API, free front-end, router, editor or terminal — set one up with a hard spend cap and a revocable key, and judge an extension or wrapper by its permissions, publisher and disclosed data use before installing it

20 · 8 min

The web app, the phone app and the desktop app

THE ONE THING TO KEEP

The website, the phone app and the desktop app are different products sharing one account, with the web usually ahead on features and app-store subscriptions usually costing more than the same plan bought on the website.

Same account, three different products

One subscription usually gets you a website, a phone app and sometimes a desktop app. People assume these are the same thing in different windows. They are not, and the differences decide which one you should be doing which work in.

The website is almost always ahead. New features ship there first, sometimes months before the mobile version. If a capability you read about is missing, check the web before concluding you do not have it.

The phone app is best at capture and worst at work. Dictation, photographs, quick questions while standing somewhere — excellent. Long documents, careful editing, comparing two answers side by side — painful, and people blame the model for what is really a 6-inch screen.

The desktop app buys three specific things, where it exists: a global keyboard shortcut that opens it over whatever you are doing, the ability to see your screen or a selected window, and local file access. Whether those are worth installing anything depends entirely on whether you would use the shortcut.

Differences that catch people out

File handling. Upload limits and accepted formats sometimes differ between web and mobile. A spreadsheet the website accepts may be refused on the phone.

Same account, two different products

Do it on the website	Do it on the phone
A long document you will read carefully	A question while you are away from a desk
Comparing two answers side by side	Dictation, and spoken conversation
Anything you will edit afterwards	Photographing a page or a whiteboard
New features, which usually land here first	Quick capture you will tidy up later
Buying the plan: the app store pads the price	Installed from the browser, not the app store

One subscription, and the website is usually months ahead on features. Neither works offline: every one of these is a thin client talking to a server, which is why the only genuinely offline route is a model on your own machine.

Voice. The phone apps have the better voice modes, and for many people this is the reason the phone app exists. Continuous conversation, interruption, hands-free.

Background work. Long-running tasks — deep research, agent runs — usually continue on the server and appear in your history everywhere. But the mobile app may not notify you, and a phone that sleeps can drop a streaming reply mid-answer. If a task takes minutes, start it on the machine you will still be sitting at.

History and memory sync. Usually shared, occasionally not, and features like projects or custom instructions sometimes only exist on the web. Check before you build a habit on the phone that will not survive.

Offline. None of them work offline. Every one is a thin client talking to a server, which is worth stating because the desktop app in particular feels like software running on your machine, and it is not. The only genuinely offline route is a local model, which module five covers.

The one that costs money quietly

Mobile subscriptions bought through an app store are typically more expensive than the same plan bought on the company's website, because the store takes a cut and the price is padded to cover it. The difference is often a couple of dollars a month, sometimes more, and it is the same subscription.

They are also harder to cancel from the wrong place: a plan bought in an app store is cancelled in the app store's subscription settings, not in the app, and people who cancel in the app find they are still being charged. Buy on the website, in a browser, and cancel where you bought it.

The progressive web app trick

Most of these websites can be installed as an app without an app store. In a mobile browser, open the site and choose *Add to Home Screen*; on a desktop browser, look for an install icon in the address bar. You get an icon, a window without browser furniture, and the web version's feature set.

This matters most where the official app is unavailable in your country's app store, where your phone has no room for another app, or where the app store version costs more. It works, it is free, and almost nobody knows to do it.

Choosing per task

- **Long document, careful comparison, anything you will edit** — web on the biggest screen available.
- **A question while you are away from a desk, or anything spoken** — phone.
- **Repeated quick questions during focused work** — desktop app, if the shortcut earns its place.
- **Anything confidential** — none of these; see module five.
- **Anything you want to script or repeat a thousand times** — the API, which is the next lesson.

The general point is worth more than the list. When a tool disappoints, ask whether you chose the wrong route before you decide the model is weak. A

model producing a poor answer to a document you photographed on a phone at an angle is not a model problem.

BEFORE YOU MOVE ON

Someone installs a chat product from a mobile app store, pays there, and later cancels inside the app. They keep being charged. What went wrong?

A. Store subscriptions are cancelled in the store's settings, so cancelling inside the app does nothing.

B. The cancellation applies only to the next billing period, so charges continue until the current term ends.

C. The account was subscribed twice, once through the store and once on the website, so one subscription remained.

D. App-store purchases cannot be cancelled by the user at all and require contacting the provider's support team to request a refund.

21 · 9 min

Your first API call

THE ONE THING TO KEEP

An API request is a model name, a list of role-tagged messages and a few settings, and because most providers accept the same format, moving between vendors or to a local model is usually a change of two lines.

Why this is worth thirty minutes even if you are not a programmer

The API is the route where you pay for what you use rather than a flat monthly fee, where you can pin a model version, where you choose the system prompt, and where nothing about the product layer is decided for you. For a

light user it is often a tenth of the subscription price. For anyone processing more than a handful of items it is the only sensible route.

You do not need to write software to use it. Free chat front-ends, covered two lessons from now, take a key and give you a chat window. But it helps enormously to see what one request looks like, because then the whole architecture stops being mysterious.

The request

Every major provider accepts something close to this shape. Sign up, create a key in the console, then:

```
curl https://api.openai.com/v1/chat/completions \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "gpt-4o-mini",
    "messages": [
      {"role": "system", "content": "You are terse. British
spelling."},
      {"role": "user",   "content": "Three risks of a fixed-
price contract."}
    ],
    "temperature": 0.3
  }'
```

That is the whole thing. A model name, a list of messages with roles, and a couple of settings. In Python:

```
from openai import OpenAI
client = OpenAI()                       # reads OPENAI_API_KEY
from the environment
r = client.chat.completions.create(
        model="gpt-4o-mini",
        messages=[{"role": "user", "content": "Summarise this
in 40 words: ..."}])
print(r.choices[0].message.content)
print(r.usage)                          # prompt_tokens,
completion_tokens
```

Two things in there matter more than they look. The **system** role is the layer that consumer products fill in for you and you now control. And `r.usage` prints exactly what you were charged for, which makes cost concrete rather than theoretical from the first minute.

The convention that makes models interchangeable

That request format started at OpenAI and became a de facto standard. Most providers and nearly every local runner now accept it, which means switching models is often a change of two lines:

```
client = OpenAI(base_url="https://api.anthropic.com/v1/",
api_key=A_KEY)
client = OpenAI(base_url="https://openrouter.ai/api/v1",
api_key=R_KEY)
client = OpenAI(base_url="http://localhost:11434/v1",
api_key="ollama")
```

The last line is a model running on your own laptop, answering the same code. This compatibility is the single most valuable thing about the API route, because it makes your work portable in a field where products are withdrawn regularly.

Compatibility is not perfect. Vendor-specific features — thinking budgets, caching controls, certain tool formats — usually need that vendor's own library. The common path works everywhere; the interesting features do not.

Reading the response

The reply comes back as JSON, and three fields in it are worth knowing by name. `choices[0].message.content` is the answer. `usage` is the token counts you were billed for. And `finish_reason` tells you *why* the model stopped: `stop` means it finished naturally, `length` means it hit the maximum output you allowed and was cut off mid-sentence.

That last one explains a complaint people bring to support constantly — answers that end halfway through a word. It is almost never the model failing; it

is a `max_tokens` setting somewhere, either yours or the front-end's default. Raise it, or ask for a shorter answer, and the truncation goes away.

What it does not give you

Be clear about what you are giving up, because people are sold the API as strictly better and it is not.

- **No memory.** Every request is independent. A conversation exists only because you resend the previous messages, and you pay for them again each turn.
- **No web search, no file store, no code sandbox**, unless you add them.
- **No mobile app, no support, no history.**
- **A bill instead of a price.** Which is the next lesson, and it is the one that goes wrong.

Getting started for nothing

Most providers give new accounts free credits, typically $5 to $10, and some free tiers on developer platforms are generous enough for real personal use. Several hosted providers of open models offer free tiers as well. Between them you can do this whole module without paying anything.

The condition is the usual one: free tiers frequently permit the provider to use your inputs to improve the service, where the paid API generally does not. If you are experimenting with your own text, that is a fair trade. If you are experimenting with a client's contract, it is not.

BEFORE YOU MOVE ON

A developer switches the base URL from a hosted provider to a local model and the same script keeps working, but a thinking-budget parameter they were passing is ignored. Why?

A. Local models cannot run in thinking mode, so the parameter has nothing to act on.

B. The base URL change did not carry the API key, so the request fell back to default settings.

C. The shared format covers common fields; vendor-specific controls are extensions others do not implement.

D. Parameters are validated by the client library, which silently drops any field the provider's documentation does not list.

22 · 8 min

Keys, spend caps, and the bill nobody wants

THE ONE THING TO KEEP

Set a hard monthly spend cap before your first API request, keep keys out of any file you might share, use one key per use so a leak revokes one thing, and never put a key anywhere a user's device can read it.

The failure that hurts

A student pushes a small project to a public code repository. The API key is in a configuration file. Automated scanners find keys on public repositories within minutes — this is a well-documented, industrialised activity — and someone else's job now runs on that account. The first the student knows is the invoice.

This is common enough that the major providers scan public code themselves and revoke keys they find. Do not rely on being rescued.

Five rules, in order of importance

1. Set a hard spend limit on day one. Every provider console has a billing page with a maximum monthly spend and email alerts at thresholds. Set the maximum to something you could pay without difficulty — for a person learning, $5 or $10. This is the single most important step and it takes ninety seconds. Do it before your first request, not after your first surprise.

2. Never put a key in a file you might share. Keys live in environment variables or in a `.env` file that is listed in `.gitignore`.

```
export OPENAI_API_KEY="sk-..."          # in ~/.zshrc or
~/.bashrc
echo ".env" >> .gitignore               # before the first com-
mit, not after
```

Note the ordering. Adding `.env` to `.gitignore` after you have committed it does nothing; the key is already in the history, and deleting the file in a later commit does not remove it from the repository.

3. One key per use. A key for your laptop, a key for the small tool you wrote, a key for the front-end app. When one leaks, you revoke one key and everything else keeps working. A single shared key means a leak takes down everything you have.

4. Revoke rather than investigate. If you think a key may have leaked, delete it in the console and make a new one. It takes twenty seconds and costs nothing. There is no situation where waiting to be sure is the better move.

5. Check the bill in the first week. Usage dashboards show spend by day and by model. Look on day three. The gap between what people expect and what they spend is largest at the start, and it is almost always a loop that ran more times than intended.

Where the money actually goes on an API

Three patterns account for nearly every unexpected bill:

Resending the conversation. In a chat, every turn resends everything before it. A long conversation's twentieth message costs many times its first. Most front-ends let you limit how many previous messages are sent; a window of the last ten is usually plenty.

A loop with no limit. Something that retries on failure, or processes a list that is longer than you thought. Always test on ten items before running on ten thousand, and print the running cost.

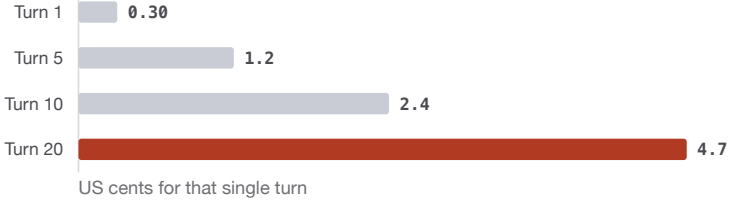
Thinking mode left on. Hidden reasoning tokens are billed at the output rate. A per-request cost can be ten times what the visible answer suggests.

Sharing a key with other people

Do not. If a colleague needs access, they create their own account, or you use a provider or router that issues separate keys with separate limits per person. A shared key means shared spending, no way to tell whose request cost what, and no way to cut off one person.

The same applies to a key inside anything a user can reach. A key in a mobile app or in a web page's JavaScript is a public key, no matter how it is obfuscated, because anyone can read what their own device received. Requests from a browser or an app have to go through a small server of yours that holds the key.

What one long conversation costs, turn by turn



Priced at three dollars per million input tokens, with roughly seven hundred tokens added per exchange. Nothing about turn twenty is harder than turn one; everything before it is going up the wire again. Capping the history at the last ten messages is the highest-value setting in any front-end.

The honest comparison with a subscription

A subscription has one genuine advantage that gets overlooked: it cannot surprise you. The price is the price, and a runaway loop just gets you rate-limited rather than billed. For somebody on a tight budget who is not confident about what they are running, that predictability is worth real money.

Use the API when you want control and low cost per use, and set the cap so that the worst case is a bad afternoon rather than a bad month.

BEFORE YOU MOVE ON

A key committed to a public repository is deleted in a later commit. Why is the key still unsafe?

A. Because provider scanners have already flagged the key and it must be re-issued through support before it can be trusted again.

B. Because the repository's cached copies on other machines will keep the old file until every clone is refreshed.

C. Because keys cannot be revoked once used, so the only remedy is closing the billing account entirely.

D. Because deleting a file in a new commit leaves the earlier commit, and its contents, in the repository history.

23 · 8 min

Building your own chat app for nothing

THE ONE THING TO KEEP

A free open-source front-end turns an API key into a chat application you control — your system prompt, your stored history, several providers and a local model in one window — at the cost of polish, built-in extras and support.

The gap the API leaves

An API key gives you models and no product. A free, open-source front-end fills that gap, and the result is a chat application you control: your system prompt, your model choice, your conversation history stored where you decide.

Four are worth knowing, all free and open source:

- **Open WebUI** — the most complete. Runs in a browser, supports multiple providers and local models at once, has document upload with retrieval, web search and multi-user accounts. Installed with one Docker command.
- **LibreChat** — similar scope, strong on multi-provider setups, plugins and organisational use.
- **Chatbox** — a simple desktop application. Install it, paste a key, use it. The easiest starting point by a distance.
- **Jan** — a desktop app built around running models locally, which also accepts remote keys.

What you gain

Every model in one window. Switch between providers mid-conversation. This is the fastest way to build the two-tool habit, because a second opinion is one dropdown away rather than another tab and another login.

Your own system prompt. Set it once — how you want to be addressed, what you do, spelling conventions, how long answers should be — and it applies everywhere, on every model, without a company's instructions competing with yours.

Conversations that are yours. Stored in a database on your machine or your server. Exportable, searchable, backed up by you, not deleted when a subscription lapses.

Cost visibility. Most of these show token counts and running spend per conversation. Seeing the number after each message changes how you use the tool within a week.

A local model beside a frontier one. The same window, one dropdown apart. Confidential material goes to the local one; everything else goes wherever is best. Nothing about this arrangement requires you to think about it after setup.

What you give up

Be honest about this, because these tools are often oversold.

- **Polish.** Voice modes, mobile apps and slick document handling lag well behind the commercial products.
- **The built-in extras.** Web search, code execution and file retrieval exist in some of these, and they are more work to set up and less reliable than the versions inside a paid product.
- **Support.** When something breaks, you are reading issues on a code-hosting site.
- **Maintenance.** Updates are yours to apply. A front-end left unpatched for a year on a public server is a genuine security problem.

Getting one running

The lowest-effort path is Chatbox: download, install, paste a key, done in five minutes. The most capable is Open WebUI, which needs Docker:

```
docker run -d -p 3000:8080 \
  -v open-webui:/app/backend/data \
  --name open-webui ghcr.io/open-webui/open-webui:main
```

Then open `localhost:3000`, create the first account, and add your provider keys in settings. On the same machine it will also find a local model served by Ollama without further configuration.

The setting worth changing on day one

Most front-ends send the entire conversation with every message, which is correct for coherence and expensive for long chats. Nearly all of them have a setting — often called context window, message limit or history depth — that caps how many previous turns get resent.

Set it to something like the last ten messages. You will barely notice the difference in quality, and on a long working conversation it can cut the cost by most of it. This is the single highest-value configuration change in any of these applications, and it is off by default in all of them.

Two cautions worth stating

Do not expose it to the internet carelessly. These tools are designed to be run on your own machine or behind a login. A copy on a public address with your API key inside is a bill waiting to happen and, if it holds documents, a data leak.

Check what any front-end sends. Most of these are honest and their code is public, which is the point of choosing open ones. Applications from app stores that offer the same convenience are frequently not: some route your requests through their own servers, where your text and sometimes your key can be read. If the app cannot explain where your request goes, assume it goes through them.

That last point generalises past this lesson. A front-end sits between you and the model and sees everything. Choosing one whose code you or somebody

else can read is not paranoia — it is the only way that question gets answered at all.

BEFORE YOU MOVE ON

Why is choosing an open-source front-end over a convenient app-store client a meaningful decision about data rather than only about features?

A. Open-source apps encrypt conversations locally, while commercial clients store them in plain text on the device.

B. A front-end sees every message, and only readable code settles where the requests actually go.

C. Open-source apps cannot transmit data anywhere, because they run entirely on the user's own machine.

D. App-store clients are required to route traffic through their publisher's infrastructure to comply with store review policies.

24 · 8 min

Routers: one key, many models

THE ONE THING TO KEEP

A router gives one key and one bill across hundreds of models, at the cost of a margin, an extra party in the data path, and — for open models — a back-end host chosen per request that may differ in quantisation and context length.

What a router is

A router sits between you and a dozen model providers. You get one account, one key, one bill, and access to hundreds of models — closed and open, from many companies — through a single request format. OpenRouter is the best known; there are several others, and some cloud platforms offer the same idea inside their own ecosystems.

For anyone comparing tools this is genuinely convenient. Testing your twelve prompts across six models normally means six sign-ups, six payment methods and six sets of documentation. Through a router it is one account and a change of model name.

What it buys you

Trying anything immediately. A new model appears and you can send it a prompt within a minute, with no new account.

Access where you could not get it. Some providers are hard to sign up for from some countries, or need a card type you do not have. A router often works where a direct account does not.

Automatic fallback. Most routers will retry a failed request on another provider serving the same model. When one company has an outage, your work continues.

One bill in one currency, with per-key spending limits you can hand to different projects or people.

Prices you can compare. Router listings show cost per million tokens across models side by side, which is the most useful pricing view available anywhere, and it is free to look at.

What it costs you

A margin. Routers make money somehow — usually a percentage on credit purchases or a small markup. It is modest, and it is real.

Another party in the data path. Your prompts pass through the router's servers before reaching the provider. That is one more privacy policy to read and one more organisation to trust, which for confidential material may be decisive.

Variable back-ends for open models. This is the subtle one and it catches people. When you request an open model like `llama-3.1-70b`, the router picks among several companies hosting it. Those hosts differ in quantisation, context length, speed and price — one may serve a 4-bit version with a 16k

window while another serves 8-bit with 128k. Same model name, measurably different behaviour, chosen for you per request.

Good routers let you pin the host, or filter by quantisation and context length, and the setting is usually off by default. If you are evaluating an open model through a router and getting inconsistent results, this is very likely why.

Feature lag. Vendor-specific capabilities — caching, thinking budgets, particular tool formats — arrive late or not at all.

Reading a router listing properly

The listing looks simple and hides three things worth checking before you trust a comparison:

1. **Input and output prices are different numbers**, often by a factor of three to five. A model that looks cheap can be expensive for long answers.
2. **The context length shown may be the host's, not the model's.** A model advertised at 128k may be served at 32k by the cheapest provider.
3. **Free entries usually mean training on your inputs, heavier rate limits, or both.** Read the note next to the word "free".

Rate limits and credits

Routers usually work on prepaid credit rather than a monthly invoice, which is quietly useful: you cannot overspend past what you have loaded, and running out is an error rather than a bill. For anyone nervous about the open-ended nature of API pricing, this is a real advantage over going direct.

The trade is that rate limits through a router are often tighter than the provider's own, because the router is sharing its own allocation across all its users. For a burst of a few hundred requests this never matters. For a sustained job it can, and it is worth a small test before you plan around it.

When to use one, and when not

Use a router for evaluation, for occasional access to many models, when direct sign-up is blocked from where you are, and for small projects where one

bill is simpler than five.

Go direct when you have settled on one provider and are using it at volume, when you need vendor-specific features, and when the material is sensitive enough that fewer parties in the path matters more than convenience.

A reasonable arrangement for most people: a router for experimenting, a direct account with whichever provider wins, and a local model for anything that must not leave the building. The router earns its margin during the deciding and stops being necessary once you have decided.

BEFORE YOU MOVE ON

Two runs of the same open model through a router give noticeably different quality and different context limits. What is the most likely cause?

A. The open model is updated frequently by its maker, so the two runs used different weights.

B. The router applied its fallback logic and substituted a different model when the first was busy.

C. The router used different hosting companies, whose deployments differ in quantisation and window.

D. Sampling variation between runs, since the router does not let you set temperature and defaults to a high value.

25 · 9 min

Where a coding assistant lives

THE ONE THING TO KEEP

Coding assistants differ mainly by what they can see and do — a file, a project, your terminal, or a cloud clone — so keep every change small, keep version control between you and any agent, and review each line before it lands.

Four shapes, and what each can see

Coding assistance is the largest paid category in this field, and the products differ far more by *where they run* than by which model they use. What the tool can see decides what it can do.

Autocomplete in the editor. Suggests the next few lines as you type. Sees the current file and usually a few related ones. Fast, cheap, and it saves real time on boilerplate. It cannot reason about your architecture because it cannot see it.

Chat in the editor. A panel beside your code. Sees whatever you attach plus, in better implementations, an index of the whole project. Good for "explain this", "write a test for this", "why is this failing".

A terminal agent. Runs as a command, reads and writes files across the project, runs your tests, and iterates. This is the shape that changes how the work feels, because it closes the loop: it makes a change, runs the test, reads the failure, tries again. It is also the shape that can do the most damage, since it edits your files.

A hosted agent. You describe a task on a website, it works in a cloud copy of your repository, and it opens a pull request. Good for well-defined, self-contained changes; poor at anything needing judgement about your codebase's conventions.

Free paths, all genuinely usable

You do not need a subscription to have any of this.

- **Continue** — an open-source extension for VS Code and JetBrains that provides autocomplete and chat against any model, including a local one.
- **Aider** — an open-source terminal agent that works with git, makes commits per change, and runs against any API or local model.
- **Cline** and similar open extensions — agentic editing inside VS Code.
- **Local models for code.** Qwen's coder models and several others run on a 16 GB laptop and handle everyday completion and small edits well.

The combination of Continue plus a local coding model is a complete, free, offline setup. It is meaningfully worse than a paid frontier assistant on hard problems and entirely adequate for the routine two thirds.

The rule that matters more than the tool

Review every line before it lands. Not because these tools are bad — they are extremely good — but because the failure mode is code that looks right and passes the test you were shown. A subtly wrong function that compiles is more expensive than one that does not.

Two habits protect you, and they cost nothing:

1. **Work in small commits.** A change you can read in two minutes is a change you can judge. An agent that rewrote forty files in one go is unreviewable, and people accept those because reviewing them is unpleasant.
2. **Keep version control between you and the agent.** Commit before letting an agent loose. Then `git diff` is your review tool and `git checkout .` is your undo. Without that, an agent that goes wrong halfway through is a genuine problem.

Permissions, which is where people get hurt

A terminal agent that can run commands can run any command. Most ask before executing, and most offer a mode that stops asking. That mode is conve-

nient and it is how somebody's test database gets dropped.

Sensible defaults: let it read freely, ask before writing, always ask before running anything that touches a network or a database. If a tool offers an allowlist of safe commands, spend ten minutes on it once. And be careful with an agent that reads issues, pull request comments or web pages, because text it reads can contain instructions aimed at it, and it has your permissions.

Choosing, briefly

- Working in one file at a time, mostly writing new code: **autocomplete** covers it.
- Understanding an unfamiliar codebase: **editor chat** with project indexing.
- Repetitive changes across many files, or test-fix loops: **a terminal agent**.
- Small, well-specified tasks you want done while you do something else: **a hosted agent**.
- No budget, or code that cannot leave your machine: **Continue or Aider with a local model**.

The model matters less than which of these you are in. A middling model in a terminal agent that can run your tests will outperform an excellent model in a chat window that cannot see your project.

BEFORE YOU MOVE ON

A terminal agent rewrites forty files in a single run and the tests pass. Why is this a worse outcome than the same work in eight small commits?

A. Because larger changes are more likely to contain conflicts when merged with other people's work.
B. Because agents produce lower-quality code when asked to change many files at once than when asked for one change at a time.
C. Because passing tests do not establish correctness, and a diff too large to read comfortably is one nobody reviews properly.
D. Because a single commit cannot be reverted without losing every change it contains, whereas small commits can be undone individually.

26 · 8 min

Extensions, wrappers and the permission dialog

THE ONE THING TO KEEP

An AI browser extension usually requests permission to read and change every page you visit, so check publisher, age, recent reviews and the data disclosure before installing, and never hand a third-party app a key you cannot cheaply revoke.

The category with the worst average quality

App stores and browser extension stores contain thousands of AI assistants. A minority are good. A large share are a thin layer over an API somebody else runs, charging a monthly fee for access you could have free, and a meaningful number are worse than that.

This is the part of the field where people lose money and data, and it catches people who are being sensible everywhere else. It catches them hardest where

the official products are hard to reach or hard to pay for, which is exactly where the need is greatest.

Reading a browser extension's permissions

When you install an extension, the browser tells you what it may do. The line that matters is this one, or wording close to it:

> ***Read and change all your data on all websites***

That is what it says. An extension with that permission can read every page you visit — your webmail, your bank's dashboard, your employer's internal systems, anything you have logged into — and modify what appears on them. Most AI extensions request it, because reading the current page is the feature.

The permission is not automatically wrong. It is what a "summarise this page" button needs. But it means the extension deserves the same scrutiny as a person you are handing your browser to, because functionally that is the transaction.

Before installing, check four things, in about three minutes:

1. **Who publishes it.** A named company with a website, or an anonymous developer account created recently.
2. **How many users, and how old.** Ten thousand users over two years is a different proposition from thirty thousand in three weeks.
3. **Recent reviews, sorted by newest.** Extensions get sold. The classic pattern is a useful extension acquired by someone else, followed by an update that inserts tracking or ad injection into a tool that already has permission to change every page.
4. **The privacy disclosure.** Stores require a data-use declaration. Read it. "Collects browsing activity" in an AI summariser means the pages go somewhere.

The wrapper business

The other pattern is simpler: an app that takes your question, sends it to a major provider's API, returns the answer, and charges you monthly for the privilege. Sometimes it adds genuine value — a good interface, a domain-specific set-up, offline queuing, a payment method that works in your country. Often it adds nothing but a price.

How to tell in two minutes:

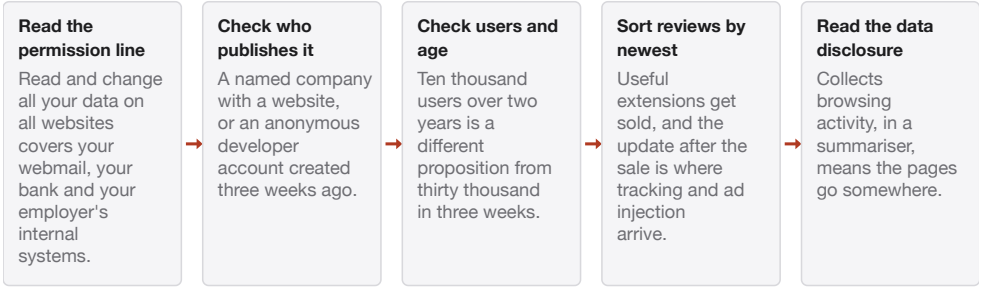
- **Does it name the model?** Honest products do. "Powered by advanced AI" means no.
- **What does it charge relative to the underlying cost?** A wrapper charging $15 a month for something that costs $1.50 of API usage is selling convenience, and you should know that is what you are buying.
- **Does it need an account and a card for a free trial?** Not disqualifying, but check the renewal terms before entering anything.
- **Is your data in the deal?** Many free wrappers are free because your conversations are the product.

The honest cases are worth naming too. In markets where the official apps are unavailable or unpayable, a local wrapper that accepts local payment methods and answers in the local language is providing real value, and paying a margin for access you otherwise would not have is a reasonable trade.

Where the risk concentrates

Anything asking for a key. A third-party app that wants your OpenAI or Anthropic key can spend it. Give it a key with a low limit that you can revoke, never your main one.

Three minutes before you install a browser extension



The permission is not automatically wrong — reading the current page is the feature. It does mean the extension deserves the scrutiny you would give a person you were handing your browser to, because functionally that is the transaction.

Anything asking to sign in with Google or Microsoft. Read the scopes on the consent screen. "See, edit, create and delete all of your Google Drive files" is a request some AI apps genuinely make, and once granted it stays granted until you revoke it in your account settings. Check that page occasionally; most people have forgotten half of what is on it.

Anything on your work machine. The extension has the same access to internal systems that you do, and installing it may breach a policy you agreed to.

The safe default is boring and effective: use the official app or the official website, add a small number of extensions you have actually checked, and treat the rest of the category as advertising.

BEFORE YOU MOVE ON

A browser extension with 30,000 users and good older reviews shows recent complaints about injected adverts. What is the most likely explanation?

A. A recent browser update broke the extension, and the adverts come from pages it now fails to clean up.
B. The users complaining have other extensions installed, and are attributing another extension's behaviour to this one.
C. It was sold or its update was compromised, and its page-changing permission now serves adverts.
D. The publisher has added an advertising tier and the complaints are from users who did not read the updated terms of service.

27 · 9 min

When the phone is the computer

THE ONE THING TO KEEP

On a phone, install web apps to the home screen rather than native apps, treat text as effectively free but photographs and voice as expensive, prefer fast models on poor connections, and use a small local model when there is no connection at all.

Designing your setup around the actual constraint

For a very large share of the people reading this, the phone is not the second device. It is the only one, it has 3 or 4 GB of RAM, storage is nearly full, and data is bought in bundles. Almost every article about choosing AI tools is written by somebody for whom none of that is true.

Three constraints matter, and each has a specific answer.

Storage: do not install anything

Every major assistant works fully in a mobile browser. Open the site, then use *Add to Home Screen* — the browser's install option. You get an icon and a full-screen window, using a few hundred kilobytes instead of the 200 to 400 MB a native app takes.

You also sidestep two other things: app-store pricing, which is usually higher for the same subscription, and app-store availability, which varies by country. The web version is often ahead on features anyway.

Data: know what a conversation actually costs

Text is small. This is the good news and it is worth being concrete, because people ration text chat as though it were video.

```
a long reply       ~= 4,000 characters  ~= 4 KB
plus page overhead ~= 20 KB per exchange, roughly
250 exchanges      ~= 5 MB
```

Five megabytes for a fortnight of heavy text use. Against that:

```
one photograph uploaded    1–4 MB   (one message = a fortnight
of text)
voice mode                 ~1 MB per minute
a video call with a model  far more
a 4 GB local model         one download, then nothing
```

The practical rules follow directly. Text is effectively free; use it without counting. Resize photographs before uploading — most phone galleries have this, and a 1024-pixel-wide image is enough for the model to read a document while being a tenth the size. Avoid voice mode on a metered connection. And if the site keeps reloading and re-sending on a poor connection, that is the real cost, not the conversation.

An unreliable connection

Long answers stream token by token, and a connection that drops mid-answer loses the answer. Two habits help: ask for shorter replies, and copy anything valuable out immediately rather than leaving it in a tab.

If you are on a poor connection, prefer the smaller and faster model even when the strong one is available. A reply that arrives in two seconds survives a connection that a forty-second thinking-mode reply does not.

Running a model on a phone

It is possible, and the honest summary is that it works better than people expect and is more limited than the demonstrations suggest.

PocketPal and **MLC Chat** are free apps that run small open models on Android and iOS. A 1 to 3 billion parameter model at 4-bit quantisation is roughly 700 MB to 2 GB and runs on a mid-range phone at a usable speed.

What it is good for: drafting and rewriting, summarising text you paste in, simple translation, and working with no connection at all. What it is not good for: facts about the world, anything long, anything requiring careful reasoning. It will also warm the phone and drain the battery quickly.

The genuine use case is offline. On a train, in a field, during an outage, in a place with no coverage — a 2 GB file on your phone answers questions when nothing else will.

The cheapest useful habit on a small phone

Keep one conversation per task rather than one long conversation for everything. On a phone this is not about tidiness. A conversation that has run to two hundred messages re-sends its whole history each turn, which on a metered connection and a slow processor is felt as the app becoming sluggish and the replies arriving late.

Starting a fresh conversation, with a two-line summary pasted in if you need continuity, makes the app faster and the answers better at the same time, be-

cause the model is no longer reading three hours of earlier confusion before it reaches your question.

Other routes worth knowing

In several countries, assistants are reachable over **WhatsApp** or by SMS, which matters where a browser session is expensive and messaging is zero-rated on the network. These are usually feature-limited compared with the app, and worth knowing about anyway.

Shared devices deserve one line of caution. If the phone is shared with family, conversation history is visible to whoever picks it up. Use the private browsing window, or clear conversations, or keep anything sensitive off the device entirely. This is a real consideration for a large number of people and it is rarely mentioned.

BEFORE YOU MOVE ON

Someone on a metered data bundle rations their text conversations carefully but uploads several photographs a day. What does the arithmetic say?

A. Photographs and text cost roughly the same once page overhead is counted, so the rationing is proportionate.
B. Both should be rationed, since the streamed reply to an image question is far longer than a normal text reply.
C. They have it backwards: one unresized photograph can cost as much as a fortnight of text exchanges.
D. Neither matters much, because mobile browsers compress uploads automatically before sending them.

28 · 8 min

Voice, screen readers and hands-free use

THE ONE THING TO KEEP

Route decides usability before model quality does: dictation and read-aloud are free on every device, streaming replies interact badly with screen readers, and a plain-text API client is accessible by construction when a product's interface is not.

A route decision, not an afterthought

For a lot of people the question is not which model is best but which route they can actually use: someone with limited vision, with a motor impairment, with dyslexia, with a repetitive strain injury, or simply someone whose hands are busy. The tools differ enormously here, and the differences are rarely mentioned in any comparison.

Speaking instead of typing

Two different things get called voice input and they are worth separating.

Dictation turns speech into text in the input box, which you then edit and send. Accurate, cheap, works everywhere, and it is the better option for anything long or precise. Your phone's built-in dictation is free and often as good as anything in the app.

Voice mode is a spoken conversation: you talk, it talks back, you can interrupt. Genuinely good for thinking aloud and for hands-free use. Worse for anything you need to keep, because the reply is speech and reviewing it means reading a transcript anyway.

Accent handling has improved and remains uneven. Speech recognition trained mostly on a few accents does measurably worse on others, and this is one of the clearest places where the technology works better for some people

than others. Test it in your own accent, in your own language, before you build a habit on it.

Free paths are strong. **Whisper** transcribes dozens of languages and runs on an ordinary laptop; **whisper.cpp** makes it fast on a CPU. Dictating into any text field and pasting is available to everyone at no cost, without a subscription that includes voice.

Having it read back to you

Text-to-speech is built into every operating system and is free: VoiceOver on Apple devices, Narrator or the Read Aloud feature in Windows, Select-to-Speak on Android, Orca on Linux. For better voices at no cost, **Piper** runs locally and produces natural speech offline in many languages.

This matters for a use most people miss: having long AI output read to you while you do something else is a very good way to review it. Errors that the eye skims over are noticeable when heard.

Screen readers and chat interfaces

Streaming replies are genuinely awkward with a screen reader. Text appears token by token, and depending on how the page announces updates, the reader may repeat the growing answer, interrupt itself, or read nothing until the end.

Products differ, and this is worth testing before committing:

- Does the answer get announced once when complete, rather than continuously?
- Are the controls — model picker, attach, regenerate, stop — reachable and labelled with the keyboard?
- Is the conversation a sensible reading structure, or a wall of unlabelled containers?
- Can you turn streaming off? Some products offer this and it usually improves screen-reader use immediately.

Where the official app is poor, the free front-ends from earlier in this module are worth trying, since they are ordinary web applications and some are more standards-compliant than the commercial products. Where none is workable, the API route with a plain text client is fully accessible by construction.

Hands-free and keyboard-only

Keyboard-only use is mostly a matter of shortcuts: a global hotkey to open the assistant, and a reliable way to send without reaching for a mouse. Desktop apps generally do this better than websites.

For heavier needs, **Talon** and **Dragon** drive a computer by voice; the free and open **Serenade** does something similar for coding. These are general accessibility tools rather than AI tools, and they combine well with any of the routes here.

Reading and writing difficulties

One use deserves naming on its own. For people with dyslexia, these tools are unusually good at a specific job: taking text you have written and fixing spelling, punctuation and structure without changing what you meant. That is a task with a checkable answer, it does not require the model to know anything, and even a small local model does it well.

The same works in reverse. Asking for a long document to be restated in plain sentences, or as a numbered list, makes material accessible that was written for somebody else. Both of these are ordinary chat, need no special product, and are worth more to some readers than every feature comparison in this course.

The point worth making plainly

An assistant you can speak to and be read back by is, for some people, the difference between doing a piece of work independently and needing help with it. That is a larger effect than any difference in model quality this course discusses, and it is a reason to test the route before the model — the best model in an

interface you cannot navigate is worth less to you than a middling one in an interface you can.

BEFORE YOU MOVE ON

A screen-reader user finds that a chat product re-announces the whole answer repeatedly as it streams in. What is the most direct fix to try first?

A. Switch to a smaller, faster model so the answer completes before the reader can repeat it.

B. Turn streaming off if the product offers it, so the reply is announced once when complete.

C. Use voice mode instead, since a spoken reply avoids the screen reader entirely.

D. Ask for shorter answers, since a brief reply produces fewer update announcements while it is being generated.

CASE STUDY · MODULE 3

Forty machines and one key

A polytechnic in Coimbatore deciding how a hundred and eighty second-year students will reach an assistant for a semester project

The department of computer applications has forty lab machines with eight gigabytes of memory each, a shared Windows image, and students in three batches. Nearly all the students also have phones: mid-range Android, storage close to full, data bought in bundles. The head of department, Mr Selvaraj, had forty-five thousand rupees for the semester and a project brief that required every student to use an assistant and to record what it cost them.

Four routes were put on the table at a staff meeting and the arithmetic disposed of one immediately. A hundred and eighty personal subscriptions at twenty dollars a month is not a semester budget, it is a department's annual one.

The second was a single department key on an open-source front-end installed on the lab server, with every student reaching it from a browser. One account to set up, one bill, works on the lab machines and from the students' phones on the college network. Mr Selvaraj wanted this and wanted it on Monday.

Vimala, the lab technician, objected on two grounds, and her objections were the substance of the meeting. A single shared key means there is no way to tell whose request cost what, so a runaway spend is a mystery as well as a bill. And a student loop with a retry on failure, running against two thousand items on a Thursday night, can empty the semester's budget before anybody sees an email.

Her alternative was a router with a separate key per student and a spending limit on each. That answers both objections exactly. It also means creating and later revoking a hundred and eighty keys, in a semester where she was already running three other labs, and it means somebody handling the prepaid balance every few weeks.

The fourth route was to tell students to use free tiers on their own devices, which costs nothing and produces no comparable work, because the tools cap at different points, some of the apps are not listed for older Android versions, and several students did not have three hundred megabytes free to install anything.

Then a student asked whether he could simply install a browser extension that put an assistant beside every page. The department had no policy. The extension's permission line said it could read and change all data on all websites, and the lab machines are used to log into the college's internal portal, where attendance and marks live.

Vimala had a second objection to the shared key that nobody expected, and it was the better one. The project brief required each student to record what their work had cost. With one key and no attribution, the only number available is the department's total, which teaches nothing. The students would be writing down a figure somebody else had calculated for them, which is the opposite of the exercise.

A lecturer, Mrs Anitha, raised the case nobody had planned for. Eleven of the hundred and eighty students are in the evening batch and come to the lab twice a week. Whatever route the department chose had to work from a phone on a metered connection at home, or those eleven would do the project by borrowing somebody's laptop on a Saturday.

That put a constraint on the answer that the budget had not. Voice and image uploads are expensive on a bundle; text is not. A photograph is a megabyte or more and a whole afternoon of text conversation is a few. Whatever was chosen had to be usable as text on a poor connection, and anything that streamed long replies over a connection that dropped would lose the answer along with the data spent on it.

Mr Selvaraj's position was that Monday mattered: a route that exists imperfectly on Monday teaches more than a perfect one that arrives in week four. Vimala's position was that the first runaway loop would end the experiment for everybody, and that the person who would be asked to explain it to the principal was her.

What would you have run, and what would you have spent the budget's first rupee on?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

They took the shared key and closed Vimala's objections a different way. The front-end supports its own user accounts, so each student signed in individually while a single key sat behind all of them. That gave attribution — whose conversation, how many tokens — without a hundred and eighty keys to create and revoke.

The key's cap was set in the provider's console at four thousand rupees for the month, under a tenth of the budget, with a written rule that it would not be raised mid-month. Two alert thresholds were set on the same page.

The front-end's history setting, which by default resends the whole conversation on every message, was capped at the last ten. Spend in the second week was less than half the first week's for the same number of students, and nobody noticed a difference in the answers.

On the phones, the assistants were installed from the browser to the home screen rather than from the app store, which cost a few hundred kilobytes instead of three hundred megabytes and sidestepped the two students whose Android version was not supported.

Extensions were prohibited on lab machines, and the permission line was read aloud in the class that asked, which did more than the prohibition. In week six a retry loop did run away against two thousand items. It stopped at four thousand rupees on a Thursday night, and the log named the account in under a minute.

A shared key with per-user accounts in the front-end is not the same as per-user keys. What does it give you, and what does it not?

It gives attribution: the front-end records which account sent which conversation and how many tokens it used, so a spike has a name against it. It does not give isolation. There is still one credential, so a leak revokes access for everyone rather than for one person, one student cannot be cut off without changing the key, and the spending limit is shared — the cap protects the department's budget but not one student's share of it from being consumed by another's loop.

Capping the history at the last ten messages halved the spend in a week. Why is that the highest-value setting in any of these front-ends?

Because every turn of a conversation resends everything before it, so the twentieth message of a long chat costs many times the first. Most of the money in conversational use is earlier turns being re-read, not new text being written. Capping the resent history removes that cost with almost no effect on quality for ordinary work, and it is switched off by default in every one of these applications.

Why did the department explain the extension's permission rather than simply prohibiting extensions?

Because the prohibition only covers the lab. Students will install extensions on their own machines, and a rule they do not understand does not travel with them. The permission line — read and change all your data on all websites — is what makes the risk concrete: an extension with it can read every page the user is logged into, including a bank dashboard or the college portal, and extensions are sold and updated after installation. Teaching the check (publisher, age, recent reviews, data disclosure) is the part that is still useful after the semester.

MODULE 3 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 Why is the same subscription usually more expensive bought inside a phone app than on the company's website?

A. The app store takes a cut and the price is padded to cover it.

B. The mobile version serves a larger model, which costs the company more to run.

C. Mobile requests are routed through more infrastructure, and providers price by region of access.

D. The app bundles voice mode, which is billed separately on the website and included on mobile.

2 An API answer stops in the middle of a word. What is the usual cause?

A. The model lost the thread of the answer and stopped, which indicates a model too small for the task.

B. The response hit the maximum output length set by your request or by the client's default.

C. The context window filled up while the answer was still being written.

D. The connection dropped mid-stream, which truncates a reply without reporting any error.

3 You committed a file containing an API key, then added it to the ignore list and deleted the file in a later commit. What is the state of the key?

A. Safe, because the file no longer exists in the working tree and is now ignored.

B. Safe once the repository is made private, which removes earlier commits from public view.

C. Still exposed, because it remains in the repository's history.

D. Still exposed until the next push, after which the deletion propagates and the history is rewritten.

4 Which single setting in a free chat front-end most reduces the cost of long working conversations?

A. Lowering the temperature, because fewer surprising tokens means fewer tokens in total.

B. Turning off streaming, which stops you being billed for tokens you never read.

C. Reducing the maximum output length, which caps what the model is allowed to write in each reply.

D. Capping how many previous messages are resent with each new one.

5 You evaluate an open model through a router and get inconsistent results across one morning. What is the most likely cause?

A. The router picks a different back-end host per request, and hosts differ in quantisation and context length.

B. The router retries failed requests behind the scenes, and a retried answer is generated at a higher temperature.

C. The router caches answers, so some replies are older versions of the model's output.

D. Open models are inherently less consistent than closed ones because their weights are public.

6 A browser extension requests permission to read and change all your data on all websites. What does approving it grant?

A. Access to the page you press the button on, for as long as the button is pressed.

B. The ability to read and alter every page you are signed into, including mail and banking.

C. Access to your browsing history, but not to the contents of the pages themselves.

D. Access limited to the sites listed on the store page, which the store enforces on the publisher's behalf.

MODULE 4

What it actually costs

Money decides more tool choices than quality does, and almost nobody does the arithmetic. This block does it: what a token is and why output costs several times input, how to cost a job from a twenty-item pilot before running fifty thousand, the caching and batching discounts most people never switch on, how to route easy work to cheap models, the three billing shapes and their different risks, the costs that never reach the invoice, and what to do when the card does not work or the price is simply out of reach.

> BY THE END OF THIS MODULE YOU CAN
>
> Estimate the cost of a job from a measured sample before running it, cut that cost with prefix caching, batching and model routing without lowering quality, choose between a subscription, prepaid credit and pay-as-you-go for your own pattern of use, and set an enforced cap so the worst case is bounded

29 · 8 min

The free tiers, and their traps

THE ONE THING TO KEEP

A free tier shows you a product's floor, not its ceiling, and it will downgrade you without saying so.

What "free" usually contains

Every major assistant has a free tier, and they are more generous than most people expect. A typical one gives you:

- A limited number of messages on the strong model, then an automatic switch to a smaller one.
- Smaller limits on file size, uploads per day, and images.
- Web search, sometimes, and often fewer searches.
- Little or no memory across conversations.
- Your conversations used to improve the model, unless you change a setting or live somewhere the law says otherwise.

That is enough for a great deal of real work. A student writing essays, a shopkeeper drafting supplier emails, a nurse checking her English phrasing — none of these people need to pay anything.

Six traps

1. The silent downgrade. You use up your allowance of the strong model, the product moves you to a smaller one, and the only sign is that answers get shallower. People conclude "it got worse this week." It did not. You did. Look for the model name in the interface before you form an opinion.

2. Judging a company by its free tier. The free tier is the cheapest model the company is willing to serve at scale. It is not what the product can do. If

you are deciding whether to pay, that decision cannot be made on free-tier output alone.

3. Rolling windows, not calendar days. Limits usually reset on a rolling window — five hours from your first message, not at midnight. Plan the heavy task for the start of a window.

4. Free trials with a card attached. A one-month trial that renews at full price is a normal business practice and a real cost if you forget. Set a reminder for two days before it renews, the day you sign up.

5. Free API credits that expire. Several providers hand new developers credits worth $5 to $300. They expire, often in 30 to 90 days, and they are not refundable. Fine, as long as you know.

6. Wrappers charging for something free. App stores and browser extension stores are full of apps that charge a monthly fee to pass your question to a model you could have used at no cost. This trap catches the most people in markets where the official apps are hard to reach or pay for. Before paying anyone, check whose model is underneath.

Free things genuinely worth knowing about

- **Developer free tiers.** Google's AI Studio has historically offered a free quota on its API that is generous enough for small projects. Others offer free credits. These come with the usual condition: free usually means your data can be used to improve the service.
- **Open models on your own machine.** No account, no limit, no message cap, and nothing leaves your laptop. Two lessons from now.
- **Cheap hosted open models.** Several companies serve Llama, Qwen, Mistral and DeepSeek at a fraction of frontier prices, some with a free tier.
- **School and university access.** Many institutions have licences their students do not know about. Ask.

Getting a lot from free

Run two or three free tiers side by side. When one caps you, move to the next. Save your strongest-model messages for the task that needs them — the hard analysis, not the quick rewrite. Batch small questions into one message rather than spending five turns on them.

A person who is deliberate about this gets most of the value of a paid plan for nothing. The next lesson is about the cases where that stops being true.

BEFORE YOU MOVE ON

Kwame has been using a free assistant all morning. By afternoon, answers in fresh conversations are noticeably shallower on the same kind of question. What is most likely happening?

A. Servers are busier in the afternoon, so the model gets less compute per answer.

B. His conversation grew long, so the model has forgotten the earlier context and is guessing.

C. He crossed a usage cap and the product switched him to a smaller model without making it obvious.

D. The model was updated during the day and the new version is worse.

30 · 8 min

When paying is worth it

THE ONE THING TO KEEP

Pay when a cap is blocking work you would have done anyway, not to own the best model.

Say the number out loud

The standard consumer plan for the main assistants has settled around $20 a month. Coding assistants sit lower, around $10 to $20. Business and team plans run roughly $25 to $60 per person.

Twenty dollars is about 1,700 rupees, 30,000 naira, 400 pesos, 320,000 rupiah. In much of the world that is a real decision, not a rounding error, and anyone who talks about it as trivially cheap is not talking to you. So let us be exact about when it earns its keep.

Four honest reasons to pay

You keep hitting caps. This is measurable. For two weeks, note every time you are blocked or downgraded mid-task. If it happens more than three or four times a week, you are paying in interrupted work already.

You need a specific paid-only capability. Larger file uploads. Running code. Connecting to your documents. A much longer context. Project workspaces that keep instructions across chats. Look at the feature you actually need and check whether it is behind the wall.

The time is worth more than the money. If a tool saves you three hours a month and three hours of your time is worth more than the fee, the arithmetic is finished. For a freelancer billing anywhere, this is usually a short conversation. For a student on no income it may genuinely not be.

You need better terms for your data. Business and team tiers generally come with a contract that your content is not used for training. If you handle client or patient information, that clause may be the entire reason to upgrade — not the model.

Four reasons not to

You use it twice a week. Free covers you. Nothing else to decide.

You are buying "the best model." The gap between the best paid model and a good free one is small on most everyday work and invisible on some of it. Run your five prompts from the last lesson before you assume otherwise.

You are paying for three at once. This is common and mostly waste. Pay for one, use two others on their free tiers as second opinions. You lose almost nothing.

The feature you want is free somewhere else. Image generation, transcription, translation and coding help all have capable free or cheap options from different companies. Paying one vendor for everything is convenient, not optimal.

The option most people skip

Almost every provider sells access by the token through an API, and you can use it from free open-source chat apps that run in your browser or on your desktop. You pay for what you use.

The rough shape of it: small and mid-sized models cost well under a dollar per million input tokens. Frontier models cost several dollars per million input tokens and more for output. A million tokens is roughly 750,000 words.

So a person who asks fifteen moderate questions a day, mostly on a mid-sized model, may spend $2 to $5 a month instead of $20. A person feeding whole codebases to a frontier model every day can pass $20 before the first week ends.

Two honest warnings. There is no cap unless you set one — go into the billing page and set a hard monthly limit on the first day. And you are now responsi-

ble for the product layer: no memory, no polished mobile app, no support, unless the client you chose provides it.

A test you can run this month

Keep a note for two weeks. Every time you are blocked by a limit, or you want a feature you do not have, write one line. At the end, count.

More than eight lines: pay for the plan. Two to eight: try the API route first. Under two: stay free, and spend the money on something else.

And cancel without ceremony when it stops earning its place. Your conversation history stays; you drop to the free tier; you can come back next month. There is no loyalty being tested here.

BEFORE YOU MOVE ON

Fatima pays about $20 a month for three different assistants and wants to cut cost without losing much. What is the strongest reason she can drop two of them?

A. All frontier assistants now produce essentially the same output, so the extra two add nothing.

B. The free tiers of the two she drops will still cover the occasional second opinion she uses them for.

C. Concentrating on one will let its memory learn her preferences and cover what the others did better.

D. API access is always cheaper, so she should cancel all three and pay per token instead.

31 · 8 min

The unit you are billed in

THE ONE THING TO KEEP

You are billed per token, output usually costs three to five times input, hidden thinking counts as output, and non-Latin scripts can take several times more tokens for identical content.

Why the bill is not in words

Models do not read letters or words. Text is cut into **tokens** — chunks that are often a whole common word, sometimes a fragment, sometimes a single character. `unbelievable` might become `un`, `bel`, `iev`, `able`. A space usually travels with the word that follows it. Punctuation is often its own token.

The rules of thumb, for English:

```
1 token   ~= 4 characters  ~= 0.75 words
1,000 tokens ~= 750 words  ~= 1.5 pages of prose
1,000,000 tokens ~= a 3,000-page book
```

These are averages and they hold well for ordinary English prose. They hold badly for code, for tables of numbers, and for anything that is not English.

Input and output are priced differently

This is the part people miss and it changes decisions.

Output tokens typically cost **three to five times** input tokens. The reason is mechanical: input can be processed in parallel in one pass, while each output token has to be generated one at a time, each one requiring a full pass through the model.

```
a model at $3 per million in, $15 per million out

reading a 50-page report (30k tokens) and writing 300 words
(400 tokens)
   input   30,000 / 1,000,000 x $3   = $0.090
   output     400 / 1,000,000 x $15  = $0.006
                                       -------
                                       $0.096

writing a 5,000-word article from a short brief
   input      200 / 1,000,000 x $3   = $0.0006
   output   6,700 / 1,000,000 x $15  = $0.100
```

Almost the same price, opposite shapes. If your work is reading long things and answering briefly, you are buying input and should care about input price. If it is generating long documents, output price is your bill. Comparison pages that quote one number are quoting the wrong one half the time.

Thinking tokens are output tokens

A thinking-mode reply generates hidden reasoning before the visible answer, and that reasoning is billed at the output rate. A 300-word answer preceded by 4,000 words of thinking costs roughly fourteen times the answer alone. This is the single largest surprise on API bills, and it is invisible in the chat window.

Non-English text costs more

Tokenisers are fitted mostly to Latin-script text. The same meaning in Devanagari, Bangla, Thai, Amharic or Georgian can take two to four times as many tokens, because the tokeniser has to fall back on shorter pieces.

Three consequences at once: you pay several times more for the same content, you fit proportionally less into the context window, and generation takes longer because there are more tokens to produce. Chinese and Japanese sit in between — dense in characters, and many characters are single tokens.

This is a real and rarely acknowledged inequity in how these systems are priced, and it is worth knowing about rather than being puzzled by a bill.

Counting them yourself

You do not have to estimate. Every API response reports exactly what you used:

```
r = client.chat.completions.create(model=M, messages=msgs)
print(r.usage.prompt_tokens, r.usage.completion_tokens)
```

For counting before you send, the free **tiktoken** library counts for OpenAI-family tokenisers, and **transformers** loads the tokeniser for any open model:

```
import tiktoken
enc = tiktoken.get_encoding("o200k_base")
len(enc.encode(open("contract.txt").read()))     # exact, be-
fore you spend
```

Several providers also offer a free token-counting endpoint. Run your own text through one once, in your own language. The number tells you more about your future bills than any pricing page.

What a context window costs you per turn

One arithmetic worth doing once. A model with a 200,000-token window, filled, costs 200,000 input tokens on every single request that uses it. At $3 per million that is $0.60 a message, and a twenty-message conversation over that document is $12 before the model has written a word.

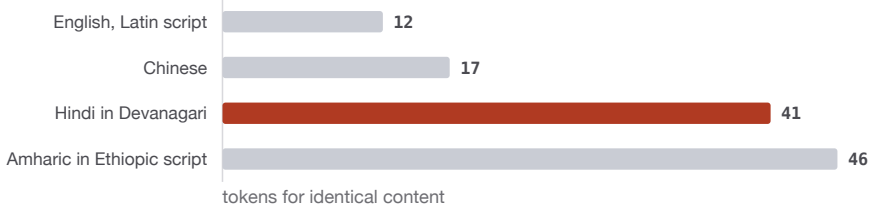
This is why long-context work is where people's API bills go, and why the fix is almost never a cheaper model. It is sending less: extract the relevant sections, start fresh conversations, and let prefix caching cover the part that genuinely has to be resent every time.

The one to hold on to

Cost is driven by three things in this order: **how much you send**, **how much you ask it to write**, and **which model you send it to**. Not by how

hard the question is. A one-line question with a 300-page attachment costs a hundred times a difficult question typed on its own, and that ordering surprises almost everyone the first time they see their own numbers.

The same sentence, four scripts



Tokenisers are fitted mostly to Latin-script text, so everything else falls back on shorter pieces. Three consequences at once: you pay several times more for identical content, you fit proportionally less into the window, and generation is slower because there are more tokens to produce.

BEFORE YOU MOVE ON

Two jobs on the same model: reading a 50-page report and replying in 300 words, or writing a 5,000-word article from a two-line brief. Why do they cost about the same?

A. Both consume roughly the same total number of tokens once input and output are added together.

B. The big input is billed at the cheap rate; the long output at the rate several times higher.

C. Providers price by request rather than strictly by volume, which flattens costs across differently shaped jobs.

D. Long inputs are compressed by the provider before processing, so a 50-page report is billed as far fewer tokens.

32 · 8 min

Costing a job before you run it

THE ONE THING TO KEEP

Run twenty real items, record actual token usage, multiply and add a third for retries — and run the same test on a small and a large model so the price of the extra accuracy is a number rather than an assumption.

Measure twenty, multiply by the rest

The reliable method has three steps and takes fifteen minutes. It is worth doing before anything that runs more than a few hundred times, and it converts an anxious guess into a number.

1. Run twenty real items. Not made-up ones — real ones, including a long one and a short one. Record the token usage for each.

```
tot_in = tot_out = 0
for item in sample_of_20:
    r = client.chat.completions.create(model=M,
messages=build(item))
    tot_in  += r.usage.prompt_tokens
    tot_out += r.usage.completion_tokens
print(tot_in/20, tot_out/20)      # average per item
```

2. Multiply.

```
per item:   1,850 in, 240 out
50,000 items:
    input   92,500,000 tokens
    output  12,000,000 tokens
at $0.15 / $0.60 per million:
    $13.88 + $7.20 = $21.08
at $3.00 / $15.00 per million:
    $277.50 + $180.00 = $457.50
```

3. Add a third. Retries after failures, items that need a second pass, the run you abandon halfway and restart. A third is a realistic allowance; anything under a fifth is optimism.

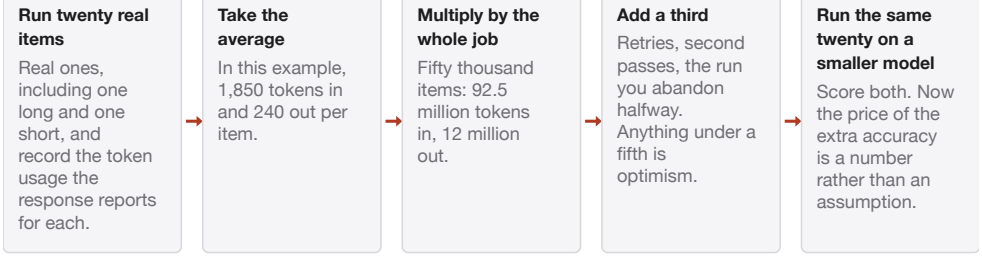
Where the estimate goes wrong

The long tail. Twenty random items rarely include the outlier. If some of your documents are ten times longer than the average, and you have a thousand of them, they dominate the bill. Look at the *distribution* of your input sizes, not the mean, and if it has a tail, sample from the tail deliberately.

Output length is not what you asked for. "Be brief" is not a limit. If the average output in your sample is 900 tokens when you expected 200, either the instruction is being ignored or your idea of brief is not the model's. Set `max_tokens` and see how often it truncates; that tells you the real distribution.

The conversation you forgot is resent. If the job is conversational rather than one-shot, every turn resends everything before it. A ten-turn conversation does not cost ten times a single turn — it costs closer to fifty times, because the input grows each turn.

Costing a job in fifteen minutes



Precision is not the point. Knowing whether the answer is twenty dollars or five hundred is, and the per-item figure — twelve paise an invoice, a fifth of a cent a ticket — is the number that ends most arguments about whether to do this at all.

Failures cost money. A request that returns a bad answer was still billed. A retry loop with three attempts triples the worst cases.

The comparison worth running

Do the twenty-item test on two model sizes, and score the outputs. This is the highest-value fifteen minutes in this whole module, because it converts "the big model is probably safer" into a number:

```
small model:  $21 for the job,  17/20 outputs acceptable
large model: $458 for the job,  19/20 outputs acceptable

$437 for three items out of twenty
->  or:  small model + hand-check the 3 flagged as uncertain
```

Very often the honest conclusion is that the small model plus a review step is better and cheaper than the large model unreviewed, because the large model's failures are the ones nobody catches.

When the job is a subscription

The same arithmetic works in reverse. If you are on a $20 plan, ask what it would have cost through an API. Take a week of your actual conversations, estimate the tokens — most front-ends will show you, and a rough count of words divided by 0.75 is close enough — and price it.

People who do this split into two groups. Light chat users find their real usage would cost $2 to $4 a month, and the subscription is buying convenience and features rather than compute. Heavy users of long documents and thinking modes find they would spend far more than $20, and the plan is a bargain. Both are useful things to know, and neither is guessable from the outside.

Estimating without any code

If you are not going to write a script, the same method works by hand. Take five real items, run them in a chat front-end that shows token counts — most of the free ones do — and note the numbers. Where nothing shows counts, divide the character count of your input by four and the words of the output by 0.75.

That estimate will be out by perhaps 20%, which is far better than the order-of-magnitude error you would make by guessing. Add the third for retries as before, and you have a number good enough to decide with. Precision is not the point; knowing whether the answer is five dollars or five hundred is.

The number that ends most arguments

Before a meeting about whether to use AI for a task, work out the per-item cost and write it on one line: *"12 paise per invoice"*, *"$0.004 per ticket"*, *"$1.20 per contract review"*. Almost every disagreement about whether this is worth doing dissolves once that number sits next to the cost of doing it the current way.

BEFORE YOU MOVE ON

A twenty-item pilot predicts $21 for a 50,000-item job, but the real run costs $190. The prompt and model were unchanged. What is the most likely cause?

A. The provider applied surge pricing during a period of high demand across the platform.

B. Token prices rose between the pilot and the run, which is common given how quickly this field moves.

C. The sample of twenty missed the long tail, and a minority of much larger inputs dominated the total.

D. The model produced longer outputs at scale because later items in a batch receive less attention per item.

33 · 8 min

The two discounts almost nobody uses

THE ONE THING TO KEEP

Put every stable block at the front of a request so prompt caching can match the prefix, send anything that can wait through the batch route at around half price, and cut what you send and what you ask for before you consider a cheaper model.

Prompt caching

If you send the same long block of text repeatedly — a system prompt with detailed instructions, a policy document, a code file, a knowledge base — the provider can store its processed form and charge a fraction to reuse it. Discounts are typically 50% to 90% on the cached portion, and some providers apply it automatically while others need a marker in the request.

The mechanism decides how you use it. The cache matches on an **exact prefix**: the request has to begin with byte-identical text. So the rule is to put everything stable at the front and everything variable at the back.

```
[ long instructions        ]  <- identical every time, cached
[ reference document       ]  <- identical every time, cached
[ today's date, user's name]  <- CHANGES: breaks the cache from
here on
[ the question             ]

move the date and the name to the END, after the question,
and the whole block above stays cached
```

That single reordering is worth a large fraction of the bill on any workload with a fixed preamble, and it is the most common cache mistake: one variable field near the top invalidates everything after it.

Caches expire — commonly a few minutes of inactivity, extendable at a price on some providers. So caching pays when requests come in bursts and does nothing for one request an hour.

Batch processing

Most providers offer a batch route: you upload many requests as a file, they are processed within a stated window — usually up to 24 hours — and you collect the results. The discount is generally around 50%.

Order the request so the cache can match

Long instructions	Byte-identical every time. Cached.
Reference document	Byte-identical every time. Cached.
The question	Varies. Everything from here down is charged in full, which is correct and unavoidable.
Today's date, the user's name	Move these below the question. One variable field near the top invalidates every cached block after it.

The cache matches on an exact prefix, so every stable block goes at the front and everything variable at the back. Discounts run from a half to nine tenths on the matched portion, and the commonest mistake is a date sitting two lines from the top.

This is the right route for anything that does not need an answer now: classifying an archive, generating descriptions for a catalogue, summarising last month's tickets, translating a documentation set. Half price for patience.

```
50,000 items, live:   $458
50,000 items, batch:  $229
```

Two practical notes. Results come back unordered, so include your own identifier with each request. And the window is a maximum, not a promise of speed — most batches finish much sooner, but you cannot rely on it, so do not put a batch job in front of a deadline.

The discounts that need nothing from the provider

Stop resending the conversation. In a chat front-end, cap the history at the last ten messages. Most of the cost of a long conversation is the earlier turns being re-read every time.

Ask for shorter answers. Output is the expensive side. A model told to answer in 100 words instead of 500 costs a fifth as much for the generation, and for most questions the 100-word version is the one you wanted.

Send less. People attach a whole document when a section would do. Extraction with `pdftotext` and a few lines of slicing costs nothing and can cut a request by 90%.

Turn thinking off when it does not help. Covered in module one; it belongs on this list because it is the biggest single lever.

Deduplicate. If the same question is asked repeatedly by different people, store answers in a table keyed by the question. A dictionary lookup costs nothing and often serves a third of the traffic in customer-facing work.

What none of this fixes

Two costs are immune to all of it. A model that has to read a genuinely long document to answer a genuinely global question about it — is anything inconsistent, what is the overall argument — must see the whole thing, and no amount of caching, batching or routing avoids paying for that once.

And an answer that has to be right cannot be made cheaper by being generated more cheaply. Where the cost of an error dominates, the correct spending decision is usually to pay for the better model and the review step, and to look for savings on the routine work that surrounds it instead.

The ordering that matters

If you want to cut a bill, do these in this order, because the first three cost nothing and the last one costs quality:

1. Send less and ask for less.

2. Cache the stable prefix, batch what can wait.
3. Route easy items to a smaller model.
4. Only then consider accepting worse output.

Most people go straight to step four by switching to a cheaper model everywhere, discover the quality drop, and conclude the work cannot be done cheaply. The first two steps are usually worth more than the model change, and they cost you nothing at all.

BEFORE YOU MOVE ON

A workload puts the current date at the top of an otherwise identical 8,000-token preamble, and prompt caching saves almost nothing. Why?

A. The preamble is too long to cache, since providers cap the size of a cacheable block well below 8,000 tokens.
B. Caching only applies to system messages, and the date placed at the top pushed the instructions into the user message.
C. Dates are treated as volatile content and excluded from caching by the provider's content rules.
D. The cache matches on an exact prefix, so a value that changes daily invalidates everything after it.

34 · 8 min

Spending money where it matters

THE ONE THING TO KEEP

A cascade — cheap model everywhere, expensive model only on flagged cases — commonly cuts cost by four fifths at equal or better quality, provided you have measured on a hundred items which cases the cheap model actually gets wrong.

The shape production systems actually have

People assume the systems they admire run the best available model on every request. Very few do. The common design is a **cascade**: a cheap model handles everything, and only the cases it cannot settle go to an expensive one.

```
1,000 items
  -> small model handles all 1,000            $0.40
  -> 120 flagged uncertain go to large model  $2.60
                                              -----
                                              $3.00
  versus all 1,000 on the large model:       $21.60
```

Eighty-six per cent saved, with the hard cases still getting the expensive treatment. This is not a compromise design. On many tasks it also scores *better* than the large model alone, because the escalation step forces a second look at exactly the items most likely to be wrong.

Deciding what escalates

You need a signal for "this one is difficult", and there are four that work without any special infrastructure.

Ask for a confidence word. Have the small model end with `CONFIDENT` or `UNSURE`. It is not calibrated and it is not a probability, and it still correlates

usefully with correctness — well enough to route on. Check the correlation on fifty items before trusting it.

Rules you already know. Some cases are structurally hard: a document over thirty pages, a ticket from a top-tier customer, an invoice above a threshold, a message in a language the small model handles poorly. Route those up without asking anything.

Disagreement. Run the small model twice at a non-zero temperature. If the two answers differ, escalate. Cheap, crude, and effective on classification.

Cost of being wrong. The most important one, and it is a business decision rather than a technical one. Anything where a mistake costs money, safety or a customer goes to the expensive model regardless of how confident anything is.

The measurement that makes it honest

A cascade is only correct if the small model is genuinely adequate on the easy cases, and the only way to know is to check.

Take 100 items. Run both models. Compare. You will get four groups, and the interesting one is the fourth:

- Both right: the cascade is safe here.
- Both wrong: the model is not the problem; the prompt or the task is.
- Large right, small wrong: this is what you are paying the large model for. How many, and do your escalation rules catch them?
- **Small right, large wrong.** This group is always bigger than people expect. Large models are not uniformly better; they are better on average, and they overthink simple cases.

If your escalation rules catch most of the third group, the cascade is working. If they do not, tighten the rules rather than abandoning the design.

Where cascades are the wrong idea

When latency matters more than money. Two model calls take twice as long. For anything a person is waiting on, that can be the wrong trade.

When the volume is small. A hundred items a day at a cent each is a dollar. Engineering a cascade to save 80% of a dollar is not a good use of an afternoon.

When you cannot measure. A cascade without the 100-item comparison is a guess that quietly ships worse work. If you are not going to check, use the model you trust.

A hundred items, run through both models

	Large model right	Large model wrong
Small model right	71 the cascade is safe here	9 small right, large wrong
nall model wrong	14 escalation has to catch these	6 the prompt or the task, not the model

The fourteen in the lower left are what you are paying the large model for, and your escalation rules have to catch them. The nine in the top right are the group people forget: large models are better on average rather than uniformly, and they overthink simple cases.

The cheapest escalation of all

There is a version of the third group that costs nothing to catch: you. In work where a person is reading the output anyway — drafting, correspondence, review queues — the escalation rule can simply be that you notice. Run everything on the cheap model, and when an answer looks thin, ask the expensive one the same question.

That is a cascade with a human as the confidence signal, and on small volumes it beats any automated version, because you are far better calibrated about your own work than a model reporting on its own certainty.

The version for a person, not a system

You do not need code for any of this. The habit is the same:

- Default to the cheap, fast model for everything routine — rewriting, summarising, formatting, ordinary questions.

- Move to the strong model when the answer matters, when the question is genuinely hard, or when the cheap one has given you something you doubt.
- Notice when you last used the expensive model out of habit rather than need.

People who work this way spend a fraction of what people who leave the strongest model selected all day spend, get answers faster for most of their work, and do not lose anything. The cheap model is not a compromise for two thirds of what anybody actually does.

BEFORE YOU MOVE ON

Comparing a small and a large model on 100 items, a team finds a group where the small model is right and the large one is wrong. What does that group mean for the design?

A. It is measurement noise from sampling, and running both models at temperature zero would eliminate the group.

B. It shows the large model needs a better prompt, since a stronger model should dominate a weaker one on every item.

C. It indicates the sample of 100 is too small, and a larger comparison would show the large model ahead on every category.

D. The models fail on different items, so escalating everything would add errors the small one avoids.

35 · 7 min

Subscriptions, credits and pay-as-you-go

THE ONE THING TO KEEP

Subscriptions cannot surprise you and are easy to underuse, prepaid credits fail safely by running out, and pay-as-you-go is the most efficient and the only shape that can produce a bill you did not expect.

Three shapes, three different risks

A subscription. A fixed monthly price for capped use. Predictable, and the cap is enforced by refusal rather than by billing. You cannot overspend. You can easily underuse — the commonest waste in this field is three subscriptions where one is used.

Prepaid credits. You load an amount and it depletes. Common on routers and some providers. Predictable in the direction that matters, because running out is an error rather than an invoice, and it is the right shape for anyone who is nervous about open-ended pricing.

Pay-as-you-go. You are invoiced for what you used. The most efficient and the only one that can surprise you. Always combined with a hard cap you set yourself.

What each is genuinely best for

Subscriptions suit steady, conversational, human-paced use where the product features — memory, projects, mobile apps, file handling, voice — are much of the value. If you use a chat assistant most days and never write code, this is your shape and the arithmetic rarely favours anything else.

Credits suit experimenting, occasional heavy bursts, and anybody who cannot risk a bill. They also suit gifts and shared budgets: you can give somebody £10 of access without giving them your card.

Pay-as-you-go suits automation, volume, and light use. A person asking a handful of questions a day usually spends a few dollars a month. The same person on a subscription spends twenty.

Things that catch people

Seat licences. Team plans charge per person per month, and they usually charge for the seat whether or not the person uses it. A team of twelve where four people use the tool is paying three times what it needs to. Review the seat list quarterly; nobody ever does this and it is often the largest single saving available to a small organisation.

Annual commitments. A discount of 15% to 20% for paying yearly, in a field where the right tool changes every six months. For a business with a settled workflow, reasonable. For an individual still deciding, it converts a reversible choice into an irreversible one for the sake of three dollars a month.

Minimum commitments and tiers. Enterprise agreements sometimes come with a spend floor. If you commit to $2,000 a month and use $600, you paid $2,000.

Credits that expire. Free promotional credits usually expire in 30 to 90 days. Purchased credits sometimes expire too — check before loading a large amount.

Overage rates. Some plans include an allowance and then charge per unit beyond it, occasionally at a rate worse than pay-as-you-go. Find that number before you rely on the plan.

Auto-renewal after a trial. Set a calendar reminder two days before, on the day you sign up. This is the single most reliable way people lose money in this category, and it is entirely preventable.

Reading the plan page properly

Three lines on a pricing page carry most of the risk, and none of them is the headline number. Find the allowance — messages, requests, credits, or "usage limits apply" with no figure at all, which is itself worth noting. Find the rate

beyond the allowance. Find the cancellation terms, including whether cancelling stops access immediately or at the end of the term.

If the page will not tell you the allowance in a number, that is a deliberate choice by the seller, and it means you cannot compare the plan with anything. Treat vagueness there as a price in itself.

Mixing them, which is what most people should do

There is no rule that says pick one. A common and sensible arrangement:

- One subscription for the tool you use daily, chosen on features and fit.
- A small prepaid credit balance on a router, for testing anything new and for the second opinion.
- Free tiers of two other products, for occasional comparison.
- A local model for confidential and offline work.

Total cost: the subscription, plus perhaps two dollars a month, plus nothing. That covers everything this course discusses, and it is a smaller monthly bill than most people currently carry across three overlapping subscriptions they forgot to cancel.

The review that pays for itself

Once a quarter, open the billing pages of everything you pay for and answer one question each: *did I use this last month?* Not *might I use it* — did I. Cancel what you did not. Everything here can be restarted in two minutes, nothing is lost by pausing, and there is no loyalty being tested.

BEFORE YOU MOVE ON

A twelve-person team pays for twelve seats and four people use the tool. Why is this different from an individual underusing a subscription?

A. It is the same waste, and the fix in both cases is to move to pay-as-you-go pricing instead.
B. Team plans usually pool usage across seats, so unused seats subsidise the heavy users and nothing is wasted.
C. It is multiplied by eight empty seats and stays invisible, since nobody sees a bill as theirs.
D. The team plan includes features the individual plan lacks, so the extra seats are buying capability rather than access.

36 · 8 min

The costs that are not on the invoice

THE ONE THING TO KEEP

The invoice is the small number: verification time, rework, learning and automation bias decide whether a tool pays, and the returns are largest exactly where you can check an answer faster than you could produce it.

The invoice is the small number

A tool that costs $20 a month can easily cost you far more than $20 a month. The costs that do not appear on any invoice are real, they are often larger, and they are the reason two people with the same subscription have completely different experiences of whether it was worth it.

Verification time

This is the big one. If a tool produces a draft in thirty seconds that would have taken you twenty minutes, but you then spend twenty-five minutes checking it

because you cannot afford to be wrong, the tool has cost you five minutes.

Whether verification is expensive depends on a property of the task, not of the model: **how long does it take to check an answer compared with producing it?**

- **Cheap to check.** Code that either runs or does not. Arithmetic you can recompute. A translation you can read. A summary of a document in front of you. Here the tool is nearly pure gain.
- **Expensive to check.** A legal citation, a factual claim about a subject you do not know, a figure taken from a long report, a recommendation whose consequences appear months later. Here the verification can cost more than the work.

The practical consequence: these tools give the largest genuine returns on tasks where you can verify quickly, and the largest illusory returns where you cannot. Notice which kind you are doing.

Rework and the confident wrong answer

A wrong answer you catch costs you the time to catch it. A wrong answer you do not catch costs whatever it costs — a supplier paid twice, a clause misread, a patient given the wrong advice, a customer told something untrue.

You cannot price this directly, but you can price it roughly by asking what the last mistake in this area cost, and how often it would have to happen to wipe out the time saved. For work where being wrong is expensive, that arithmetic frequently says use the tool for drafting and never for deciding.

Learning and switching

Getting genuinely good with a tool takes ten to twenty hours spread over a few weeks. That is a real cost, paid once, and it is why switching tools every month is expensive even when each individual tool is free. You are permanently in the beginner phase.

It is also why the six-month review cadence is right. Frequent enough to notice a real change, infrequent enough that you keep the skill you built.

Cognitive costs, stated honestly

Two of these are documented and worth knowing about.

Skill maintenance. If you stop doing a thing yourself, you get worse at it. This matters where the skill is the point — a student learning to write, a junior developer learning to debug — and matters much less where it is not, such as formatting a spreadsheet. The evidence on this is still developing and the effect size is genuinely contested. What is not contested is the mechanism: practice maintains ability, and outsourcing practice removes it.

Automation bias. People accept a machine's answer more readily than a colleague's, and check it less. This is a well-established finding across decades of research into other automated systems, and there is no reason to think it does not apply here. It is worth designing around: decide in advance which categories of answer you always verify, so the decision is not made in the moment when the answer looks convincing.

The costs that go the other way

Being fair, some hidden costs are negative — the tool saves you things that never appear on an invoice either. The email you were dreading that now takes four minutes. The document in a language you read slowly. The code you did not have to ask a busy colleague about. The blank page you did not have to face. For a great many people these are the actual value, and they are just as invisible to the accountant as the verification time.

Putting a number on it

For two weeks, keep one line per day: *minutes saved, minutes spent checking, one mistake nearly made*. It is ninety seconds a day and it produces a real answer.

Most people find the tool is worth substantially more than they pay on the tasks they can check, and worth much less than they thought on the tasks they cannot. That is a useful thing to know about your own work, and no review will tell you.

BEFORE YOU MOVE ON

Two colleagues use the same assistant. One saves several hours a week; the other finds it costs time. Their tasks differ in one property. Which?

A. How complex their tasks are, since the model handles simple work well and struggles with complex work.
B. How long their prompts are, since the one saving time has learned to give more context.
C. How long checking an answer takes against producing it, which decides if verification eats the saving.
D. How much of the work is drafting rather than deciding, since drafting is what these tools are designed for.

37 · 8 min

Paying for it from where you live

THE ONE THING TO KEEP

The sticker price is not what you pay — card fees, digital-services tax and local levies can add a quarter — and where a card or the service itself is unavailable, free tiers, prepaid API credit, hosted open models and a local model cover almost everything.

The part the pricing page does not mention

Most writing about AI pricing assumes a card that works, a currency the seller quotes in, and a country the service is offered in. For a large share of the world at least one of those is false, and the practical question is not *is it worth $20* but *can I pay at all, and what does $20 actually cost me*.

What $20 really costs you

Four things sit between the sticker price and your bank statement.

The exchange rate, which is fine, and **the card's foreign transaction fee**, which is typically 1% to 3% and is not. On a $20 monthly subscription that is up to $7 a year for nothing.

Tax. Many countries require the seller to add GST, VAT or an equivalent on digital services, commonly 15% to 20%. It is often not shown until checkout, and it is not optional.

Local levies. Some countries add a tax on outbound foreign currency payments. In India, for instance, remittance and card transactions abroad have carried collection-at-source charges under some thresholds and rules that change; this is exactly the kind of thing to check for your own country in the current year rather than trusting anything written here.

So a $20 plan can land at closer to $25 by the time it reaches you. Worth knowing before you decide it is affordable, and worth knowing when comparing against a local alternative quoted in your own currency.

When the card is refused

Common and rarely explained. The usual causes, in rough order of frequency:

- The card does not permit international online transactions. Most banks let you turn this on, sometimes only in the app, sometimes for a limited window.
- The card is a domestic-network card that is not accepted internationally.
- The provider does not operate in your country at all, and the refusal is geographic rather than financial.
- The bank blocks it as suspicious and will clear it if you call.

Routes that work when a direct card does not: prepaid virtual cards from services that operate in your region, a regional payment provider that resells access, or a router that accepts payment methods a direct provider does not. Each adds a margin and each is legitimate.

Be careful about two things. Anything asking you to misrepresent where you are may breach the terms you agreed to, and losing the account also loses

whatever is in it. And "cheap accounts" sold on marketplaces are almost always shared, resold or stolen, and they stop working.

Regional pricing, and where it exists

Some software is priced by region. In this field it is uncommon: the major assistant subscriptions are mostly one dollar price worldwide, which means a plan costing 1% of a monthly income in one country costs 15% in another for the same product.

Where regional pricing does exist it is usually on local products, or on annual plans through a local reseller. It is worth ten minutes of checking, and it is worth not expecting.

What to do when the plan is genuinely unaffordable

This is a real situation for many readers and it deserves a straight answer rather than a shrug.

- **Free tiers of two or three products, used deliberately.** Covered earlier in this module; for most everyday work this is genuinely sufficient.
- **The API with a $5 credit.** For light use this lasts months, and the arithmetic is in this module. Prepaid, so it cannot overrun.
- **Hosted open models.** Several providers serve Llama, Qwen and Mistral at a fraction of frontier prices, some with a free tier.
- **A local model.** No account, no card, no monthly cost, and no ongoing dependence on anybody's pricing decisions. On a 16 GB laptop this is a genuinely capable daily tool.
- **Institutional access.** Universities, colleges, employers and some libraries have licences their members do not know about. Ask, specifically, rather than assuming.
- **Shared household use.** One subscription used by a family is against most terms of service for simultaneous use; a single account genuinely shared within a household is a grey area you should read the terms on rather than take advice about here.

The honest summary: paying is convenient, and not paying is workable. Nothing in this course requires a subscription, and the person who thinks carefully with free tools will get more out of them than the person who pays and does not.

BEFORE YOU MOVE ON

A $20 plan arrives as roughly $25 on the statement. Which explanation fits, and what does it change?

A. The provider applied a regional surcharge for countries outside its main markets.

B. Foreign transaction fees, digital-services tax and any local levy are added on top of the quoted price.

C. The first month included a setup charge that will not recur.

D. The exchange rate moved between purchase and settlement, which typically accounts for differences of this size.

---

38 · 7 min

A budget that actually holds

THE ONE THING TO KEEP

Write down current spend, set an enforced hard cap with alerts on every pay-as-you-go account, and review every subscription quarterly against what you actually used rather than what you might use.

Three numbers, written down

A budget in your head is not a budget. Write three numbers where you will see them:

1. **What you spend now, monthly, across everything.** Include the subscription you forgot.

2. **The hard cap on every pay-as-you-go account**, set in the provider's console.
3. **The date you review it.**

That is the whole system. It is unglamorous and it works, and its main virtue is that it turns an open-ended commitment into a bounded one.

Making the cap real

A cap in a provider console is enforced by them, which is what makes it different from an intention. Set it below what you can comfortably pay, not at it — the point is that the worst case is annoying rather than damaging.

Set the alert thresholds too, usually at 50% and 80%. An email at 50% on the fourth of the month tells you something is wrong while it is still cheap to find out.

Where a provider offers prepaid credit instead of invoicing, prefer it. Running out is the safest possible failure: the work stops, nothing is owed, and you decide whether to add more.

The four things that break budgets

A loop. Something ran more times than intended. This is by far the most common cause of a surprising bill and it is why you test on ten items, print the running total, and only then run the rest.

A key someone else is using. Covered in module three. Check the usage dashboard when spend does not match your memory of what you did; a pattern of requests at three in the morning is a leaked key, not a mistake.

Thinking mode left on by default. Ten times the token cost for tasks that did not need it.

Subscriptions that accumulated. Three assistants, a coding tool, an image service, a transcription service. Each one was reasonable when you added it. Together they are more than most people would knowingly agree to spend, which is exactly why the quarterly review exists.

The quarterly review

Twenty minutes, four times a year. Open every billing page and ask two questions per item:

- **Did I use this last month?** Not *might I* — did I.
- **What would happen if I cancelled it today?**

Cancel anything where the answers are "no" and "nothing". Everything here restarts in two minutes and your history generally survives. There is no penalty for pausing and no relationship to protect.

While you are there, check whether the price has changed. Prices in this field move in both directions, and providers are not obliged to make an increase prominent.

When the free tier is the right final answer

This module has covered the arithmetic of paying. It should end by saying plainly that for a large number of readers, the correct outcome of that arithmetic is to pay nothing.

If you use an assistant a few times a week, for drafting, rewriting, explaining and ordinary questions, the free tiers of two products cover it completely. You will hit a cap occasionally, you will switch to the other one, and you will lose nothing that matters. Adding a local model for confidential work costs nothing but disk space.

That is not a compromise position or a starter arrangement. It is the right answer for that pattern of use, and the fact that everyone in this industry has an interest in telling you otherwise is a reason to be more confident about it rather than less.

After a bill you did not expect

It happens, and the response is the same each time. Open the usage dashboard and sort by day, then by model. The cause is nearly always visible within a minute: a spike on one date, a model you did not mean to use, or steady traffic at hours you were asleep.

Then do three things in order. Revoke the key, because it costs nothing and rules out the worst explanation. Lower the cap, so the same mistake cannot repeat while you work out what happened. And if the charge was genuinely a provider error or an obvious runaway on a new account, ask support — several providers do issue credits in those cases, and asking politely once is free.

The one-line version

Cap what can run away, review what accumulates, and be honest twice a year about what you actually used. Three numbers and a date will keep you inside your budget more reliably than any amount of careful reading about which plan offers the best value.

BEFORE YOU MOVE ON

Why is prepaid credit generally a safer arrangement than invoiced pay-as-you-go with a hard cap set in the console?

A. Running out stops the work and leaves nothing owed; a cap acts after the spending has happened.

B. Caps are advisory on most providers and only trigger an email rather than stopping requests.

C. Prepaid credit is usually offered at a discount, so the same work costs less than it would when invoiced.

D. Prepaid accounts are not linked to a payment card, so a leaked key cannot be used to make further purchases.

CASE STUDY · MODULE 4

Forty thousand articles and a launch date

A regional newspaper in Kochi pricing the tagging of its own archive four weeks before a new website goes live

The archive is forty-one thousand articles from 1998 onwards, about sixty per cent in Malayalam and the rest in English. The new site needs three things from each: three tags from a fixed list of sixty, a forty-word standfirst, and a flag on anything naming a living person, which routes the article to a legal read before it is republished.

Thomas, the editor, wanted the whole archive run through the strongest model available. His reasoning was not vanity. The archive is the paper's only asset that a competitor cannot buy, and a bad tag is a piece of the paper's history that nobody will ever find again.

Fathima, who is the entire technical department, ran twenty real articles first and wrote down what they actually used: about eighteen hundred tokens in and two hundred and forty out. On the frontier model that priced the job at a little under five hundred dollars. On the small model in the same family it was thirty-one. Thomas looked at both numbers and said the difference was worth it.

Then Fathima checked her sample and found she had taken the twenty articles from the top of a list, which sorted by date. Nineteen were from the last two years and four were in Malayalam. The archive is sixty per cent Malayalam, and Malayalam in its own script takes roughly three times as many tokens as the same content in English.

She re-sampled, stratified by language and by length. The average came out at about three thousand four hundred tokens in and four hundred and twenty out — nearly double her first estimate. The frontier model now priced at about six hundred and seventy-six dollars before retries, about nine hundred with the third she adds for them: a little over seventy-five thousand rupees. The small model came to forty-two dollars all in.

So the disagreement got sharper rather than easier. Thomas wanted the expensive run live, where he could watch it and stop it. Fathima wanted a cascade — the small model on everything, the expensive one only on flagged cases — and she wanted the escalation pass sent through the batch route at half price, which meant results arriving within a day rather than immediately and coming back unordered.

The launch was in four weeks. A prompt fault discovered on day three of a batch run is three days.

Thomas made an argument that deserved more respect than Fathima initially gave it. Seventy-five thousand rupees is a fortnight of a junior sub-editor's salary, and the paper had spent more than that on worse things. If the expensive model meant not having to think about the tagging again, it was cheap. What he objected to was not the saving but the number of moving parts: two models, an escalation rule, a batch window and a confidence word, each of which could be the thing that went wrong on a Thursday.

Fathima's answer was that she was not proposing to guess which items the small model could handle. She wanted to measure it on a hundred articles, against tags a person had written by hand, and to design the escalation from what that comparison showed rather than from an opinion about model sizes. Without that measurement a cascade is a guess that quietly ships worse work, and she said so.

The chief sub-editor, who would have to write those hundred sets of tags, wanted to know what happened to his afternoon if the answer came back saying the small model was fine. Fathima's reply was that the hundred tags were not wasted either way: they were the only record the paper would have of what correct looked like, and the same hundred would be the check the next time anybody touched the prompt.

And there was a category neither of them wanted to route by confidence. A missed living-person flag is not a worse tag. It is a legal read that never happened, on an article about to be republished on a public website with the paper's name on it.

What would you have measured before spending anything, and where would you have refused to let the cheap model decide?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

Fathima ran a hundred articles through both models and compared them against tags the chief sub-editor wrote by hand. Seventy-one were right on both. Six were wrong on both, which said the prompt or the task was at fault and not the model. Fifteen were right on the large model and wrong on the small one, mostly long analysis pieces and Malayalam articles where a person was referred to by an honorific rather than a name. And eight were right on the small model and wrong on the large one, almost all of them because the large model had chosen a more precise category that was not among the sixty.

The cascade was built from that. The small model ran everything. Anything over twelve hundred words, anything in which the small model set the living-person flag, and anything where it ended its answer with the word UNSURE went to the large model — about twelve per cent of the archive. The living-person category escalated regardless of confidence, because the cost of being wrong there is not a bad tag.

They ran two hundred articles live on the first morning, found the prompt returning four tags instead of three on Malayalam pieces, fixed it, and only then sent the bulk. The escalation pass went through the batch route overnight. The final bill was a hundred and sixty-four dollars, about fourteen thousand rupees.

The legal reader still found eleven articles the flag had missed, all short Malayalam pieces where the person appeared only by title. A free keyword pass over a list of three hundred local titles caught nine of them.

Fathima's first twenty items came from the top of a date-sorted list. What did that do to the estimate, and what is the general rule?

It understated the job by nearly half, because recent articles are shorter and the sample was overwhelmingly English while the archive is mostly Malayalam, which costs roughly three times as many tokens for the same content. The rule is to look at the distribution of your inputs rather than the mean, and to sample deliberately from the parts that differ — the long tail, the other language, the awkward format. Twenty random items rarely contain the outlier, and if the outliers are numerous they dominate the bill.

The hundred-item comparison found eight articles the small model got right and the large one wrong. What causes that group, and why does it matter to the design?

Large models are better on average, not uniformly better, and they overthink constrained tasks — here by choosing a more precise category than the fixed list allowed. The group matters because it is always larger than people expect, and it is the evidence that a cascade is not a compromise: escalating only the flagged cases can score as well as or better than the large model everywhere, while costing a fraction. It also warns against assuming that a disagreement between the two models means the expensive one is right.

Why did the living-person flag go to the expensive model regardless of the confidence word?

Because the escalation signal that matters most is the cost of being wrong, which is a business decision rather than a technical one. A confidence word from a model is uncalibrated and correlates only usefully with correctness, which is adequate for routing a tag and not adequate where the failure is a legal read that never happens. Anything whose error costs money, safety or a legal exposure goes to the expensive model by rule, not by confidence.

MODULE 4 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 Your work is reading fifty-page reports and answering in about three hundred words. Which number on a pricing page decides your bill?

A. The output price, because generating tokens is the mechanically expensive half.

B. Neither, because providers charge a flat rate per request once you are on a paid tier.

C. The input price, because almost all your tokens are the document going up the wire.

D. The output price, because hidden thinking tokens are billed at that rate and cannot be switched off.

2 You measure twenty real items, multiply by fifty thousand, and get a figure. What does this module say to do next?

A. Halve it, because batch processing is available on most providers.

B. Double it, because published prices exclude tax in most countries.

C. Leave it, because an average measured on real items is already the expected value of the job.

D. Add about a third for retries, second passes and abandoned runs.

3 Your request begins with fixed instructions, then a reference document, then today's date, then the question. Why does prompt caching save you almost nothing?

A. The date changes, and the cache matches an exact prefix, so everything after it is uncached.

B. Caches expire after a few minutes, so a long preamble is never cached in practice.

C. Documents are excluded from caching because they are attachments rather than message text.

D. The question at the end is what the provider hashes, so a changing question invalidates the whole cached block.

4 You run a hundred items through a small and a large model. Which group of results is usually larger than people expect, and what does it mean?

A. Both wrong, meaning the model is too small and the whole job needs the large one.

B. Small right and large wrong, meaning large models are better on average rather than uniformly.

C. Large right and small wrong, meaning the cascade design cannot be made to work.

D. Both right, meaning the comparison was run on items too easy to distinguish the two models at all.

5 Which property of a task decides whether one of these tools pays for itself?

A. How hard the task is for a person to do without help.

B. How many tokens the task consumes per item.

C. How long it takes to check an answer compared with producing it.

D. Whether the task is one the model was specifically trained on, which determines its accuracy.

6 A free tier's answers get shallower across an afternoon and you conclude the product has become worse this week. What has probably happened?

A. The provider is throttling your connection, which shortens each reply.

B. The company has deployed a cost-saving update that shortens all answers for free users.

C. You have filled enough of the context window that the product began truncating earlier messages.

D. Your allowance on the strong model ran out and you were moved to a smaller one.

MODULE 5

Open models and your own hardware

Downloadable models are the route that costs nothing per use, keeps working when the internet does not, and cannot be deprecated. This block treats them as a purchasing decision rather than a hobby: the memory arithmetic including the cache people forget, reading a model card and its licence before spending the download, choosing a quantisation, renting an open model instead of running it, the honest cases where local is the wrong answer, and the two-model arrangement most people should end up with.

> BY THE END OF THIS MODULE YOU CAN
>
> Work out what your own machine will run including context and cache, read a model card and licence well enough to decide whether to build on it, choose a quantisation and format deliberately, compare hosted open-model providers on speed, precision, context and data terms, and set a routing rule between local and hosted work you can apply without thinking

39 · 8 min

The open models

THE ONE THING TO KEEP

Open weights means you can download and keep it; it rarely means open training data, and the licences differ.

Open weights is not open source

Meta, Mistral, Alibaba, DeepSeek, Google and others release model files you can download, run, modify and, usually, build products on. This is called open weights, and it is a genuinely different thing from the closed models behind ChatGPT, Claude and Gemini.

It is also not what most programmers mean by open source. In almost every case the training data is not published, and often neither is the training code. You get the finished numbers, not the recipe. Some projects — AI2's OLMo is the clearest example — do publish data and code, and they are the exception worth knowing about.

Licences differ in ways that matter:

- **Apache 2.0 or MIT.** Do essentially what you like, including commercially. Most Qwen releases and several DeepSeek releases sit here, as do Mistral's smaller models.
- **Community licences.** Llama's, for instance, allows commercial use but adds an acceptable-use policy and conditions — historically including a clause that very large companies must ask for separate permission, and naming requirements for derivative models. Irrelevant to you personally. Very relevant if you are building a business on it.
- **Research-only or non-commercial.** Less common now, still around. Read before you build.

The short rule: downloadable does not mean unconditional. Open the licence file. It is two pages.

The families, briefly and honestly

Llama (Meta). The largest ecosystem — most tutorials, most fine-tuned variants, best supported by every tool.

Mistral (France). Efficient models with a strong reputation for doing a lot per gigabyte, and a European hosting story that matters for data-residency rules.

Qwen (Alibaba). Unusually strong across many languages, released in a wide range of sizes from tiny to very large, mostly under permissive licences. Often the best choice for non-English work you run yourself.

DeepSeek. Pushed the cost of strong reasoning models down sharply and released weights while doing it, which forced prices down across the industry.

Gemma (Google) and **Phi** (Microsoft). Small models built to run on modest hardware and punch above their size.

Where they are genuinely competitive

As of 2026, open models are close enough to frontier quality that the difference does not show for: summarizing, classification, extraction from documents, translation between well-resourced languages, rewriting, and coding help of everyday difficulty. If your task is one of those and you run it thousands of times, open models are usually the correct answer on cost alone.

Where they still lag: the hardest reasoning, long chains of tool use where the model has to plan across many steps, and broad world knowledge at small sizes. A 7-billion-parameter model simply knows less than a very large one, and it will invent things to fill the gap.

Why you would choose one

Cost at volume. Running an open model on rented hardware, or your own, can cost a fraction of frontier API prices for the same job.

Data that never leaves. Not a promise in a policy document. A structural fact about where the computation happens.

Permanence. Hosted models get retired. A model you rely on can be deprecated with a few months' notice, and your carefully-tuned prompts break. A file on your disk runs in 2030 exactly as it runs today.

Regulation. If your sector or country requires data to stay inside a border, this may be the only route open to you.

The catch

A downloaded model arrives with no safety layer, no moderation, no support, and no one to call. You are also now responsible for hosting, updates and the product layer around it. And "open" tells you about the licence, not about quality, bias or safety — those you still have to test.

The next lesson is the cheapest way to try one: your own laptop.

BEFORE YOU MOVE ON

A small company in Manila plans to build a paid product on a downloadable model released under a 'community licence' rather than Apache 2.0. What should they check first?

A. Nothing. If the weights are downloadable, the model is open source and free for any use.

B. Whether the training data was published, since they cannot legally deploy without it.

C. Whether it runs without an internet connection, since community licences require local deployment.

D. The licence text and acceptable-use policy, which can restrict commercial use, naming, or use above a certain scale.

40 · 9 min

Running a model on your own machine

THE ONE THING TO KEEP

A local model buys privacy, offline access and zero marginal cost, and charges you capability and speed for it.

It is genuinely one command now

Install Ollama or LM Studio — both free, both work on Windows, macOS and Linux. Then, in a terminal:

```
ollama run qwen3:8b
```

It downloads a few gigabytes and gives you a prompt. That is the whole setup. LM Studio does the same with a graphical interface and a model browser, which is friendlier if terminals are not your habit. Underneath both sits llama.cpp, the project that made running these models on ordinary hardware practical.

Quantization, in one paragraph

Model weights are normally stored at 16 bits per number. Quantization stores them at 8, 5 or 4 bits instead. The file shrinks roughly in proportion, the model gets faster, and it loses a little accuracy — at 4 bits, usually a little enough that you will not notice on everyday tasks.

The useful arithmetic: at 4 bits, a model takes roughly half a byte per parameter. So an 8-billion-parameter model is about 5 GB. A 32B is about 20 GB. A 70B is about 40 GB. You need memory somewhat larger than the file, because the conversation itself takes space too.

What your machine can actually run

8 GB of RAM. Models of 3 to 4 billion parameters at 4 bits. Genuinely useful for summarizing, drafting, rewriting, and offline questions. Not for hard reasoning or code you will ship.

16 GB. 7 to 9 billion parameters, comfortably. This is where a local model becomes a daily tool rather than a demo.

32 GB, or an Apple Silicon machine with unified memory. 27 to 32 billion parameters. This is where quality gets genuinely good, and where a lot of people stop needing a subscription for routine work.

64 GB or a dedicated GPU with 24 GB. 70 billion parameters, slowly on CPU, fast on GPU.

Bigger than that means a workstation or rented cloud hardware, and at that point renting a hosted open model is usually cheaper.

Speed matters more than you expect

The number to watch is tokens per second. Below about 5, reading the answer appear is genuinely irritating. Around 15 to 20 it feels like a normal chat. Above 30 it feels instant.

On a modern laptop CPU with no GPU, a 7B model at 4 bits typically manages 5 to 10 tokens per second. On a recent Apple Silicon machine, considerably more. On a gaming GPU, several times more again. Try before you decide it is unusable, and try before you decide it is fine.

Why you might

Confidentiality that is structural. A clinic in Nairobi with patient notes. A lawyer under privilege. An accountant with client books. Nothing is sent anywhere, so no policy needs to be trusted and no breach elsewhere can expose you.

Bad or expensive internet. The model works on a train, on a plane, during an outage, and on a metered connection.

Volume. Once it is on your disk, running it 10,000 times costs electricity. If you need to classify every email in a five-year archive, this is the answer.

Learning. Running one locally makes the machinery visible — the system prompt, the context window filling up, the temperature setting, what happens when you run out of room. People who have done this reason about AI better.

Permanence. No deprecation notice will ever arrive.

Why you might not

The model is usually a year or so behind the frontier on hard tasks. It drains your battery and heats your laptop. It has no web access unless you add it, no memory unless you add it, and no polished mobile app. Setup takes an evening the first time.

For most people this is a second tool, not a replacement — the one you reach for when the material should not leave the building, or when you are offline, or when there are ten thousand of something. Which is the theme of the last lesson: not one tool, but a small set.

BEFORE YOU MOVE ON

Ana's laptop has 16 GB of RAM. She downloads a 70B model quantized to 4 bits — about 40 GB — and it either refuses to start or runs unbearably slowly. Why?

A. The weights have to sit in memory while running, and 40 GB does not fit in 16 GB, so the machine falls back to disk or gives up.

B. 4-bit quantization is too aggressive, and 70B models only work correctly at 16-bit precision.

C. 70B models require a dedicated GPU by design and cannot run on a CPU at all.

D. The download is corrupt, since a correctly quantized 70B model should be around 8 GB.

41 · 9 min

Working out what your machine will run

THE ONE THING TO KEEP

Budget memory for the weights, the KV cache that grows with your context length, and the operating system — and when a local model is too slow, cut context and cache precision before you cut model size.

The arithmetic, including the part people forget

The weights are the obvious cost. At 4-bit quantisation a model needs roughly **half a byte per parameter**, so an 8-billion-parameter model is about 4.5 GB on disk and a similar amount in memory.

The part that surprises people is the **KV cache** — the model's working memory for the conversation so far. It grows linearly with how much text is in the context, and on a long document it can rival the weights.

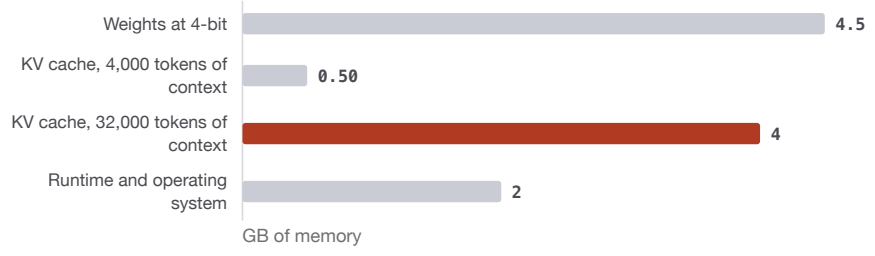
```
8B model, 4-bit weights                  ~4.5 GB
KV cache at   4,000 tokens of context    ~0.5 GB
KV cache at  32,000 tokens of context    ~4.0 GB
plus the runtime and the operating system ~2.0 GB
                                          -------
short chats:      ~7 GB       long documents: ~10.5 GB
```

So a machine that runs an 8B model beautifully for chat may swap to disk and crawl the moment you paste a fifty-page report in. That is the single most common "it worked yesterday" complaint, and the fix is not a bigger model — it is a smaller context setting, or 8-bit cache quantisation, which most runners now offer and which roughly halves that column.

What your hardware actually gives you

Ordinary laptop, 8 GB RAM. 3B to 4B models at 4-bit, short contexts. Useful for drafting, rewriting and summarising pasted text.

What an 8B model needs, beyond the weights



Short chats fit in about seven gigabytes; the same model handed a fifty-page report wants about ten and a half. That is the it-worked-yesterday complaint, and the fix is a smaller context setting or 8-bit cache quantisation rather than a smaller model.

16 GB. 7B to 9B comfortably, or a 12B at a squeeze. This is where a local model becomes a real daily tool.

32 GB. 24B to 32B at 4-bit. Quality here is genuinely good for everyday work.

Apple Silicon. Unified memory means the GPU can use most of the system RAM, which is why a Mac with 32 or 64 GB runs models that would need an expensive graphics card on a PC. Roughly 70% of total RAM is available to the model by default.

A discrete GPU. What matters is VRAM, not the card's name. 8 GB fits a 7B comfortably; 12 GB fits a 14B; 24 GB fits a 32B, or a 70B at low quantisation with reduced context. Anything that does not fit in VRAM spills into system RAM and slows down by an order of magnitude — not a little, an order of magnitude.

Measuring rather than guessing

Every runner reports speed. Look for tokens per second after a real prompt, not a one-word one.

```
ollama run qwen3:8b --verbose        # prints eval rate at the
end
```

The interpretation:

- **Under 5 tokens/sec** — painful to read along with. Fine for a job you leave running.
- **10 to 20** — comfortable conversation, roughly reading pace.
- **30 and above** — feels instant.

Watch the second number too, **prompt evaluation speed**. That is how long it takes to read your input before answering. On a CPU it can be slow enough that a fifty-page document takes a minute before the first word appears, even though generation afterwards is fine.

When it is too slow

In order of what to try, cheapest first:

1. **Reduce the context setting.** Most runners default to something large. If your work is short questions, 4,000 tokens is plenty and it frees a lot of memory.
2. **Quantise the KV cache to 8-bit.** A setting in llama.cpp, Ollama and LM Studio. Roughly halves cache memory for a negligible quality cost.
3. **Drop one quantisation level**, Q5 to Q4. Small quality cost, real speed gain.
4. **Use a smaller model.** A 4B that runs at 25 tokens/sec is more useful than an 8B that runs at 3.
5. **Offload some layers to the GPU** if you have one. Even a modest card holding half the layers helps considerably.
6. **Close the browser.** Not a joke: browsers routinely hold several gigabytes, and reclaiming them is often the difference between fitting and swapping.

The honest ceiling

There is a size above which your machine cannot go, and no setting changes it. If a model does not fit, it will either refuse to load or run at a speed that makes it unusable. That is not a failure of configuration.

At that point the choices are a smaller model, a hosted open model at a few cents an hour, or accepting that this particular task is not one your hardware does. All three are reasonable. Buying a graphics card to save a subscription is arithmetic worth doing carefully before, not after: the card costs more than several years of the subscription, and the models it fits will keep changing.

BEFORE YOU MOVE ON

A 4-bit 8B model chats smoothly on a 16 GB laptop, then crawls when a fifty-page report is pasted in. What changed?

A. The model exceeded its trained context length and fell back to a slower sliding-window attention mode.

B. The quantisation degrades on long inputs, so the model needs to reload weights at higher precision.

C. Long documents are processed twice, once for retrieval and once for generation, doubling the work.

D. The KV cache grew with the context, pushing past available memory and forcing the machine to swap.

42 · 8 min

Reading a model card before you download

THE ONE THING TO KEEP

A model card tells you the licence, the format, the context length and the languages before you spend the download — and it almost never tells you what the model was trained on, which is the question you cannot answer for a customer later.

The page that decides whether it is worth the gigabytes

Open models live mostly on Hugging Face, and each one has a model card. Most people scroll to the download button. The five minutes spent reading the card first will save you an evening.

The name, decoded. `Qwen3-8B-Instruct` is the maker's own release. `TheBloke/Something-GGUF` or `bartowski/Something-GGUF` is somebody else's conversion of a model into a runnable format — legitimate and useful, and worth knowing it is a conversion. `Someone/Model-uncensored` or `-abliterated` is a modified version with safety training removed, which is a decision you should make deliberately rather than by accident.

Base or instruct. Stated on the card. A base model completes text and will not hold a conversation. Almost everybody wants the instruct or chat version.

The licence. A field at the top. `apache-2.0` and `mit` are permissive. `llama3.1`, `gemma`, and similar named licences are custom terms with conditions. `cc-by-nc-4.0` means non-commercial. This is the next lesson but one, and it is the field most worth reading before you build anything on the model.

Size and file format. `.safetensors` is the modern format and it is the one to prefer, because it stores plain arrays of numbers. The older `.bin` weights use Python's general object-serialisation format, which reconstructs objects by running code as it loads, so a hostile file can run commands on your machine.

`.gguf` is the format for llama.cpp, Ollama and LM Studio — the one you want for running on a laptop.

Context length. Stated in the card and in the config. Note that a converted GGUF may be built with a smaller default, and that your runner may cap it lower again.

Language coverage. Good cards list the languages the model was trained on, with proportions. This is the fastest way to shortlist models for non-English work, and it is far more informative than any benchmark.

The parts that are marketing

Benchmark tables in the card. Self-reported by the maker, run under their chosen settings. Directionally useful within a family — the 32B really is better than the 8B — and not comparable across makers.

Download counts and likes. Weak signals dominated by tutorials and by whichever conversion appeared first. A model with 2 million downloads is not necessarily better than one with 40,000; it is better known.

"Comparable to GPT-4". Every release says this. Some are telling the truth for some tasks. Run your own prompts.

The three things a card usually will not tell you

What it was trained on. Almost no open-weights model publishes its training data. This means you cannot check for contamination, cannot assess bias from the source material, and cannot answer a customer who asks whether their data was in it. AI2's OLMo family is the notable exception, publishing data and code, and it is worth knowing that the exception exists.

How it behaves on your language in practice. Listed coverage is not measured quality. Test it.

Whether the safety training survived the conversion. Quantised community builds occasionally behave differently at the edges from the maker's release.

Fine-tunes and merges

A large share of what is on Hugging Face is not an original model but somebody's adjustment of one — fine-tuned on a particular kind of text, or merged from two or three others. Some are excellent and fill real gaps: a model tuned for a specific language, or for a professional domain, or for a writing style.

They also carry two risks worth knowing about. A fine-tune can lose capability elsewhere while gaining it on its target, and the card rarely measures that. And merges are produced quickly, in volume, often without evaluation of any kind. Prefer the maker's own release unless a fine-tune solves a problem you actually have, and then test it against the base model on your own prompts rather than on its card.

A five-minute checklist

1. Is it the instruct version?
2. What is the licence, and does it permit what I intend?
3. Is there a GGUF conversion, and what quantisations does it offer?
4. What context length, and does my machine have memory for it?
5. Is my language listed?
6. When was it released? A model from eighteen months ago is usually beaten by something a third of its size now.
7. `.safetensors` rather than `.bin`.

Then download the smallest quantisation that fits comfortably, try your twelve prompts, and only fetch a bigger file if the small one nearly works. A 20 GB download you abandon is an evening and, on a metered connection, real money.

BEFORE YOU MOVE ON

Why prefer a `.safetensors` file over the older `.bin` weight format?

A. Safetensors files are smaller, so the same model downloads faster and uses less disk.

B. Safetensors includes the licence and model card inside the file, making provenance verifiable.

C. The older format runs code as it loads, so a hostile file can run commands on your machine.

D. The older format cannot be quantised, so it must be converted before it will run on a laptop.

43 · 8 min

Choosing a quantisation

THE ONE THING TO KEEP

Take Q4_K_M unless you have memory to spare, prefer a smaller model at 4-bit over a larger one at 2-bit, and watch for drifting instruction-following as the sign that you have quantised too far.

The trade you are making

Weights are trained at 16-bit precision. Quantisation stores them with fewer bits — 8, 6, 5, 4, sometimes 3 or 2. The file shrinks roughly in proportion, generation speeds up because there is less memory to move, and quality degrades. The whole practical question is where on that curve to sit.

The shape of the curve is the useful thing to know, and it is not linear:

- **8-bit** — indistinguishable from the original for practical purposes. Half the size.
- **6-bit** — very close. A good choice when memory allows.

- **5-bit** — small measurable degradation, rarely noticeable in use.
- **4-bit** — the standard choice. Detectable on careful measurement, fine for everyday work. Quarter of the original size.
- **3-bit** — noticeable. Instruction-following starts to slip.
- **2-bit** — usually a false economy. A smaller model at 4-bit is almost always better than a large one at 2-bit.

That last line is the most useful rule in the lesson. Given a memory budget, prefer a smaller model at a comfortable quantisation over a larger model crushed to fit.

Reading the names

GGUF quantisations carry names like `Q4_K_M`. The parts: `Q4` is 4-bit, `_K` means the newer k-quant scheme which is better than the old one at the same size, and `_S`, `_M`, `_L` are small, medium and large variants that keep more precision in the layers that matter most.

Four bits is the standard choice because the curve is nearly flat above it and falls away sharply below. Given a fixed memory budget, a smaller model at four bits beats a larger one crushed to two, and the first sign that you have gone too far is instruction-following drifting rather than facts degrading.

`Q4_K_M` is the default recommendation and has been for a long time. If you do not want to think about it, take that one. `Q5_K_M` if you have memory to spare, `Q6_K` if you have plenty.

You will also see `IQ` prefixes — importance-matrix quants, which use a calibration pass to decide which weights to keep precise. At the same file size they are generally better than the plain equivalent, particularly at the low end where 3-bit and 2-bit become viable rather than merely possible.

The other formats, and when they apply

GGUF — for llama.cpp, Ollama, LM Studio. Runs on CPU, GPU or both, on any platform. This is what you want on a laptop.

AWQ and **GPTQ** — 4-bit formats for GPU serving through vLLM and similar. Faster than GGUF on a proper GPU, no CPU fallback. This is what a hosting provider uses.

MLX — Apple's format, for Apple Silicon. Noticeably faster than GGUF on a Mac for the same model.

bitsandbytes — quantise on the fly when loading a full-precision model in Python. Convenient for experiments, slower than a purpose-built format.

Judging it for yourself

Published perplexity numbers tell you a quantisation is 2% worse in a statistical sense. What you want to know is whether it is worse at your work, and the answer is often no.

Run four of your twelve prompts at Q4 and at Q6 and compare. On drafting, summarising and rewriting you will usually see no difference at all. On multi-step reasoning and code you sometimes will. That test takes fifteen minutes and settles it for your machine and your tasks.

One specific failure worth watching for, because it is the characteristic sign of quantising too far: the model starts ignoring parts of the instruction, or drifts out of the requested format halfway through a long answer. Facts degrade gently; instruction-following degrades noticeably.

The download itself

One practical note, because it catches people on limited connections. Model repositories hold every variant in the same place, and a naive download can fetch all of them — twenty files totalling 200 GB when you wanted one 5 GB file.

Fetch the specific quantisation by name. In Ollama, `ollama pull qwen3:8b-instruct-q4_K_M` names the variant directly. On Hugging Face, use the file browser and download the single `.gguf` you want, or pass a filename pattern to the command-line tool. Check the file size on the page before you start, and if you are on a metered connection, check it twice.

What quantisation does not do

It does not make a model know more. It does not extend the context window. It does not remove the licence terms. And it does not make an 8B model into a 70B — the ceiling on what a model can do is set by its training, and quantisation only decides how much of that survives on your hardware.

BEFORE YOU MOVE ON

With 12 GB of memory, is a 32B model at 2-bit or a 14B model at 4-bit the better choice?

A. The 14B at 4-bit: degradation accelerates below about 3 bits and costs more than the parameters gain.

B. The 32B at 2-bit, since parameter count is the dominant factor in capability and quantisation only trims precision.

C. Neither, since a 32B model cannot be loaded in 12 GB at any quantisation level.

D. The 32B at 2-bit for knowledge questions and the 14B at 4-bit for anything requiring careful instruction-following.

44 · 8 min

Renting an open model instead of running it

THE ONE THING TO KEEP

Hosted open models sit between a laptop and a frontier subscription, and the same model name from two providers can differ in speed, quantisation, context length and data terms — so check those before comparing prices.

The route between a laptop and a frontier subscription

Several companies do nothing but serve open models over an API: you send a request naming `llama-3.3-70b` or `qwen3-32b`, they run it on their hardware, you pay per token. Prices sit well below frontier models and the quality on ordinary tasks is close enough that the difference rarely shows.

This is the most under-used route in the whole field. It covers the large space between "my laptop cannot run this" and "I am paying frontier prices for summarising".

What you are comparing between providers

The same model name from two providers is not the same service. Five things differ and all of them are checkable before you commit.

Price per million tokens. Compare input and output separately.

Speed. Tokens per second varies enormously between providers running identical weights, because it depends on their hardware and serving stack. Some specialist providers are several times faster than others on the same model. If you are generating long documents, this is the difference between a workable tool and an irritating one.

Quantisation. Some serve at full precision, some at 8-bit, some at 4-bit, and many do not say. A provider serving a heavily quantised version at a lower

price is offering a different product under the same name. Ask, or look for a published statement; the good ones publish it.

Context length served. Frequently lower than the model supports. A model advertised at 128k may be offered at 32k.

Data terms. The important one. Some providers state plainly that they do not train on or retain your inputs; some have a free tier that does both; some are quiet about it. This is the question to settle before the price question, because a cheap provider that keeps your client's contract is not cheap.

Where this route wins

Bulk work. Classification, extraction, summarising, translation, tagging — the tasks where open models match frontier models and volume makes price decisive.

Predictable, moderate use. A few dollars a month rather than twenty.

Speed-critical work. The fastest hosted open models generate several hundred tokens per second, which is faster than any frontier model and changes what interactive tools feel like.

Trying a model before installing it. Cheaper than a 40 GB download you may not keep.

Data residency. Some providers let you choose the region your requests are processed in, which matters where regulation requires it.

Where it does not

The hardest tasks. Long multi-step reasoning, agentic tool use over many steps, and the deepest knowledge questions still favour frontier models. Test rather than assume; the gap has narrowed and it has not closed.

Confidentiality that must be structural. A promise in a policy is not the same as computation on your own machine. If the answer must be "the data never left the building", this route is not it.

Feature parity. No memory, no web search, no file store unless you build them.

Checking a provider in ten minutes

Before committing to one, run three things. Send the same prompt to two providers serving the same model and compare the answers side by side; large differences point at different precision. Time a 500-word generation on each and note the tokens per second. And read the data page rather than the marketing page, looking specifically for the words *retention*, *training* and *human review*.

If the third of those cannot be answered in ten minutes, that is your answer for anything sensitive. Nothing else about the provider needs checking, because the question that mattered has already been left unanswered.

A sensible arrangement

For most people who have got this far, the arrangement that works is three-way and costs very little:

- **Local model** for anything confidential and anything offline.
- **Hosted open model** for volume and for routine work, at a few dollars a month.
- **One frontier model**, subscription or API, for the hard things.

The interesting part is how rarely the third is needed once the first two are set up properly. People who make this arrangement typically find that the frontier tool handles perhaps a tenth of their requests, and that the tenth is worth paying for while the other nine tenths were not.

BEFORE YOU MOVE ON

Two providers serve the same open model at very different prices, and the cheaper one does not state its quantisation. What follows?

A. The price difference is most likely infrastructure efficiency, since quantisation choices are standardised across providers.
B. The cheaper one is running at lower precision, and the released weights make that verifiable by comparing outputs.
C. Quantisation is irrelevant to a hosted service, since the provider absorbs the memory cost rather than passing it on.
D. You may be buying a different product under the same name, so ask about precision and context before comparing on price.

45 · 8 min

The licence is a business decision

THE ONE THING TO KEEP

Read the licence rather than the word used to describe it: Apache and MIT remove a whole class of question, community licences permit commercial use with conditions, and whether any of this counts as open source is a live, unresolved dispute.

Three categories, and what each permits

Permissive — Apache 2.0, MIT. Use, modify, redistribute, sell, embed in a product. No conditions worth worrying about beyond keeping the notice. Many Qwen, Mistral and DeepSeek releases sit here. If you are building anything commercial, this is the easy category.

Custom community licences. Meta's Llama licences and Google's Gemma terms are the best-known examples. They permit commercial use, and they add conditions. Historically these have included an acceptable-use policy, a

requirement to pass the terms downstream, naming requirements for derivative models, and — in Llama's case — a clause requiring a separate agreement above a very large monthly-user threshold. For an individual or a small business these are almost never a practical obstacle. They are a real obstacle if you are a large platform, and they mean a lawyer should read them before you build a company on one.

Non-commercial and research-only. `cc-by-nc`, research licences, and evaluation-only terms. Less common than they were and still around, particularly for research releases and some specialist models. Read before you build.

Why "open source" is contested here

This is a genuine, unresolved disagreement, and it is worth understanding rather than picking a side.

The traditional definition of open source, maintained by the Open Source Initiative, requires freedom to use, study, modify and share without restrictions on field of use or on who may use it. Several widely-used model licences fail that test — usually because of acceptable-use restrictions or user thresholds — which is why the OSI and others argue those models should be called **open weights** rather than open source.

The counter-argument is that releasing weights that anyone can download, run and fine-tune delivers most of the practical freedom people care about, and that acceptable-use terms are a reasonable response to real risks.

There is a further dispute about whether a model can be open source at all when its training data is not published, since the data is arguably the source and the weights the build output. The OSI's own definition for AI systems requires detailed data information rather than the data itself, which satisfies almost nobody completely.

None of this is settled. Different jurisdictions, standards bodies and companies use the terms differently, and the disagreement is ongoing. What matters for you is practical: **read the actual licence rather than the word used to describe it.**

The other unresolved question

Alongside the definitional dispute there is a legal one, and it is worth naming as unsettled rather than glossed. Models are trained on large amounts of text and images collected from the web, and whether that training is permitted without the rights-holders' consent is being litigated and legislated differently in different countries. Some jurisdictions have text-and-data-mining exceptions; others do not; several cases are ongoing.

That uncertainty attaches to closed and open models alike, and nothing in a licence resolves it, because a licence governs what the maker permits you to do with the weights, not whether the weights were lawfully made. Anyone telling you this question is settled — in either direction — is describing their preference rather than the law. This is not legal advice; if the answer matters commercially where you are, it is a question for a lawyer in your jurisdiction.

The questions to answer before building

1. **May I use this commercially?** Stated plainly in permissive and non-commercial licences; requires reading in the custom ones.
2. **Must I attribute, and how?** Some require naming the base model in derivative names, or displaying "Built with X".
3. **Do the terms travel with it?** If you fine-tune and release, are you obliged to pass the licence on?
4. **Is there a threshold?** Some terms change above a user count.
5. **What does the acceptable-use policy prohibit?** These read like usage policies for hosted services and they bind you as an operator.
6. **What is the output's status?** Most licences do not claim ownership of what the model generates, but this is separate from whether that output is copyrightable at all, which differs by country and remains unresolved. This is not legal advice and the answer where you live may differ from the answer next door.

The practical advice

For learning, personal use, and internal tools, all three categories are usually fine and the distinction is mostly academic.

For anything you sell, embed in a product, or build a business on: prefer Apache 2.0 or MIT, because it removes an entire class of question. If a community-licensed model is genuinely better for your task, read the terms in full, keep a copy of the version you agreed to, and get advice if the numbers are large. The licences have changed between releases before, and the copy on the website today is not necessarily the one you accepted last year.

BEFORE YOU MOVE ON

A team plans to embed an open model in a product they sell, and the best-scoring candidate carries a custom community licence. What is the correct next step?

A. Read the full terms, keep a dated copy, and weigh the conditions against a permissive alternative.

B. Avoid it entirely, because a licence that fails the open-source definition cannot be relied on in a commercial product.

C. Use it, since community licences permit commercial use and the conditions apply only to very large platforms.

D. Ask the model's maker for written confirmation that the intended use is permitted before proceeding.

46 · 7 min

When running it yourself is the wrong answer

THE ONE THING TO KEEP

Local models are the right answer for confidential, offline, high-volume and permanent work, and the wrong one for hard reasoning, whole-product needs, battery life, other people depending on it, and buying hardware to avoid a subscription.

The enthusiasm and the correction

Running a model on your own hardware is genuinely satisfying and genuinely useful, and it is oversold by people who enjoy it. This lesson is the correction, written by somebody who thinks the previous ones in this module are right.

Seven cases where local is the wrong choice

You need the hardest reasoning. A model that fits your laptop is roughly a year behind the frontier on difficult multi-step problems, and the gap is largest on exactly the tasks where you would most value help.

Your time is worth more than the difference. Setup is an evening. Keeping up with new releases, formats and runner updates is an hour a month if you care. If a $20 subscription is not a strain, that arithmetic frequently favours paying.

You are buying hardware to save on subscriptions. A graphics card costs the equivalent of two to four years of a subscription, will be superseded, and only fits the model sizes it fits. If you want the card anyway, that is a different decision. As a cost-saving measure it usually is not one.

You need the whole product. Web search, file handling, memory, a mobile app, voice, sharing with colleagues. You can assemble these around a local model and it is a project, not a setting.

You are on a laptop battery. Local inference is a heavy, sustained load. It warms the machine, spins the fans, and shortens the working day considerably.

You need it to work for other people. The moment a second person depends on it, you have become the operator of a service — availability, updates, security patches, and someone to blame when it is down.

You are relying on a safety layer. A downloaded model arrives with whatever safety training its makers gave it and no product layer around it. If it will be used by children, students or the public, that absence is your responsibility to fill.

The security surface people forget

A self-hosted front-end reachable from the internet is a service you are running, and it needs the same care as any other. The failure pattern is dull and common: an instance put up quickly for convenience, left unpatched and without a login, holding both an API key and a year of documents.

If you run one, keep it on your own machine or behind authentication, keep it updated, and do not put anything on a public address without deciding deliberately to run a public service.

Where local is unambiguously right

To be balanced, the cases where nothing else will do:

- The material must not leave the building, as a matter of law, contract or professional duty.
- There is no reliable connection.
- The volume is enormous and the task is routine.
- You need the same behaviour in five years with no deprecation notice.
- You are learning how these systems work, which is a genuinely good reason on its own.
- You cannot pay, and this is the route that costs nothing after the download.

The maintenance nobody mentions

There is an ongoing cost that never appears in a setup guide. Runners update and occasionally change behaviour. Model formats evolve, and a file that worked last year may need re-downloading in a new format. Better models come out constantly, so the model you installed is quietly becoming the wrong choice. And an operating system upgrade can break a GPU driver stack in ways that take an evening to unpick.

None of this is difficult. It is an hour a month if you keep up and a lost weekend if you do not, and it is the honest reason many people who set this up enthusiastically are back on a subscription a year later. Decide whether you want that hour before you begin, rather than discovering it afterwards.

The honest summary

A local model is a very good second tool and a demanding first one. The arrangement that works for most people is the one at the end of the previous lesson: a local model for the material that requires it, something hosted for everything else, and no ideological commitment to either.

The mistake in both directions is the same. Choosing on identity — "I self-host everything" or "I only use the best model" — instead of on what the particular piece of work needs.

BEFORE YOU MOVE ON

Someone plans to buy a graphics card to avoid a $20 monthly subscription. What does the arithmetic suggest?

A. The card costs several years of subscription, will be superseded, and only fits certain model sizes.

B. It is sound if the card can run a 70B model, since that closes the quality gap with a frontier subscription.

C. It pays for itself within a year, since the card also serves other uses and the subscription is pure cost.

D. It depends mainly on electricity prices, which over a few years exceed the purchase price of the card.

47 · 8 min

A working setup with two models

THE ONE THING TO KEEP

Route work by what the material contains rather than by how hard it looks — confidential to the local model, routine and bulk to a cheap hosted one, the hard remainder to the strong one — and write the rule down once so the choice stops being a decision.

Deciding once instead of every time

The arrangement that works is not one tool. It is two or three, with a rule that decides which to use so you are not making the choice every time. The rule that works is not about difficulty — it is about **what kind of material you are holding**.

```
Does this contain something that must not leave my machine?
    yes -> local model
    no  -> is this routine work, or a lot of it?
             yes -> hosted open model or the cheap tier
             no  -> the strong model
```

Material first, difficulty second. This matters because difficulty is a judgement you get wrong under time pressure, while "is there a client name in this" is a fact you can see.

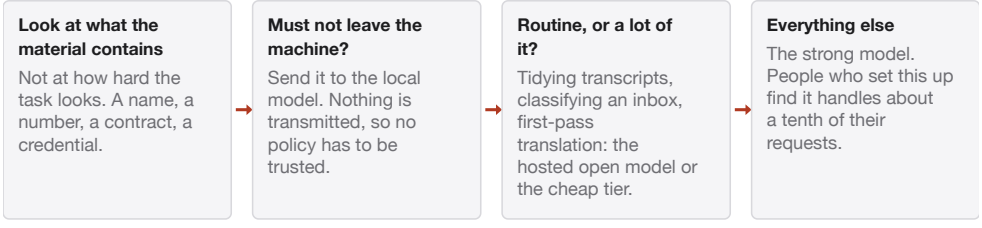
Making it a habit rather than a decision

Put the local model somewhere immediate. A desktop app, or a front-end where it is one dropdown away from everything else. If reaching it takes three steps, you will not use it on the Tuesday when it matters.

Set the default to the cheap option. Whatever you use most, set it to open on the small fast model. Reaching for the strong one should be a deliber-

ate act; sitting on it all day is how people spend a fortune and wait unnecessarily.

The routing rule, decided once



Material first, difficulty second. Difficulty is a judgement you get wrong under time pressure; whether there is a client name in the document is a fact you can see. Written down once, the choice stops being a decision.

Keep a redaction habit rather than a redaction step. Writing "the supplier" instead of the company name costs nothing, produces the same answer, and removes the decision entirely for a large share of your work.

A concrete setup, end to end

For a self-employed person handling client material, on a 16 GB laptop:

1. **Ollama** with an 8B instruct model — installed once, used for anything with a name, a number or a contract in it.
2. **A free front-end** — Chatbox or Open WebUI — with the local model and two API keys configured, so all three are in one window.
3. **A hosted open model** on prepaid credit, for bulk work: tidying transcripts, classifying an inbox, first-pass translation.
4. **One frontier model**, on a subscription or on the API, for the hard tenth.
5. **The twelve-prompt file** in a folder next to it all, and a calendar entry for September.

Total ongoing cost: one subscription, or none, plus a few dollars. Setup: one evening. This is not an advanced configuration; it is what somebody who has thought about it once ends up with.

What to check every so often

Two things drift, and both are cheap to re-check at the six-month review.

The local model has probably improved. Small open models have been getting better at a rate that makes an eighteen-month-old choice worth revisiting. A model that failed your prompts 7 to 9 last year may pass them now at the same size.

Your rule may be too cautious or not cautious enough. If everything is going to the local model and you are dissatisfied with the answers, the rule is too tight. If nothing is, you have stopped noticing what is in your documents.

A worked example of the rule in use

A physiotherapist writing up notes. The clinical note itself, with the patient's name and history, goes to the local model for tidying and structuring — that is material that must not leave the machine, and tidying prose is exactly what a small model does well.

The general question that came out of the session — what the current evidence says about a particular rehabilitation protocol — goes to the strong hosted model, because it needs knowledge the local model does not have and it contains nothing identifying. And the fifty referral letters that need reformatting go to the cheap hosted model in a batch overnight.

Three tools, one afternoon's work, and the routing was decided by looking at what was in front of her rather than by thinking about model capability at all.

The point of the whole arrangement

It is not thrift, and it is not privacy purism. It is that no single tool is right for every kind of material you handle, and that the cost of having two or three ready is close to zero once they are set up.

The person with this arrangement is not choosing between privacy and capability, or between cost and quality. They have arranged things so that each piece of work goes to whichever tool suits it, and the deciding takes no

thought because the rule was written down once. That is the whole ambition of this course applied to a single desk.

BEFORE YOU MOVE ON

Why route by the material's sensitivity rather than by how difficult the task looks?

A. Because routing on difficulty requires measuring model confidence, which consumer products do not expose to users.

B. Because local models handle sensitive material more accurately, having fewer distractions in their training data.

C. Because sensitive material is usually simpler, so the local model is sufficient for it in practice.

D. Because difficulty is a judgement made under time pressure, while what a document contains is a fact you can see.

CASE STUDY · MODULE 5

A laptop, a graphics card, and a clause in the engagement letter

A chartered accountant in Surat working out how to summarise four years of a client's ledgers without the ledgers leaving her office

Nikita Shah practises alone and keeps twelve clients. The largest is a garment exporter whose engagement letter contains an ordinary clause: client records may not be disclosed to any third party without written consent. She had never thought of it as an AI clause, because it was drafted in 2019.

The work in front of her was four years of ledger narrations — about forty thousand lines, most of them abbreviated, inconsistent between the two people who had entered them, and full of names. She wanted a model to group them, flag the ones that did not match a head of account, and draft the management letter that came out of it.

Her machine is a three-year-old laptop with sixteen gigabytes of memory and no separate graphics card.

Three options were on the table. A business-tier subscription at about twenty-two hundred rupees a month buys a contract saying the content is not used for training, a data processing agreement and a retention setting. All of that is real and none of it answers the engagement letter, because a data processing agreement does not make a disclosure into something other than a disclosure. A hosted open model is cheaper and faster and is also a third party. And a machine that would run something larger locally — thirty-two gigabytes, ninety-five thousand rupees — is four years of the subscription in a year where she was also replacing a printer and had just started paying a part-time assistant.

So she installed Ollama and an eight-billion-parameter instruct model on the laptop she had. In chat it was excellent: drafting, rewriting, tidying her own notes, about fourteen tokens a second, entirely adequate.

Then she pasted a single year of narrations into it. Thirty-eight thousand tokens. The machine slowed to a crawl, the fans came on, and the first word of the answer took over a minute to appear.

She assumed the model was too big for the machine. It was not. The weights were about four and a half gigabytes and had been fine all week. What had changed was the working memory for the conversation, which grows with the length of the context and on a document of that size had come to rival the weights themselves, on top of the runtime and everything else the laptop was already holding.

The decision in front of her was whether to buy the machine now, or to split the work — local for anything carrying the client's identity, hosted for everything else — and accept that the ledger job would run in small batches overnight over several days rather than in an afternoon.

The case for buying was not only speed. Her practice is her own, there is no IT department, and she disliked the position of having the confidentiality of her largest client depend on a setting she might change by accident. A machine that could hold a larger model comfortably would let her stop thinking about the boundary altogether, which is a real thing to want.

The case against was the part that is easy to leave out of the arithmetic. Ninety-five thousand rupees is four years of the subscription she was comparing it against, and the subscription's model improves underneath her for nothing while the machine holds still. A colleague in Rajkot had bought a graphics card eighteen months earlier for exactly this reason and had since spent a lost weekend on a driver problem after an operating system upgrade, and another evening re-downloading every model he had because a file format had moved on.

There was a third option nobody had put on the table, which was to ask the client. Nikita had not considered it because she had framed the whole question as a technical one about where computation happens, when the clause in the engagement letter is about consent and consent is something a client can give. The exporter's finance director is a person she speaks to most weeks.

What would you have tried before spending ninety-five thousand rupees, and what would you have put in the letter?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

She did three things, none of which cost money.

She reduced the runner's context setting from its large default to eight thousand tokens, switched the conversation cache to eight-bit, and closed the browser, which on its own returned about two and a half gigabytes. The eight-billion model then handled the narrations in batches of two hundred lines at about twelve tokens a second, running overnight across four nights. No purchase was made.

She wrote to the client. One paragraph: which service she proposed to use for non-identifying analysis, which tier, what would and would not be sent, and a request for written consent. The reply came in two days and asked for the same paragraph in a form the exporter could show their own auditor.

And she found that the habit of writing the role rather than the name — the supplier, the customer, the director — meant most of the management-letter drafting never needed the local model at all, because nothing identifying was in it.

At her September review a four-billion model released that spring did the narration job on the same laptop at about thirty tokens a second. The case for the ninety-five thousand rupee machine was weaker in September than it had been in March, which is the usual direction.

Why did a model that worked perfectly in chat crawl on a year of narrations, and what are the two cheapest fixes?

Because the weights are not the only thing in memory. The KV cache — the model's working memory for the context so far — grows with the amount of text in front of it, and on a long document it can rival the size of the weights. The machine then swaps to disk and speed collapses. The two cheapest fixes are to lower the context

setting, which most runners default to something far larger than short questions need, and to quantise the cache to eight-bit, which roughly halves that column for a negligible quality cost. Both are settings, not purchases.

A no-training commitment and a data processing agreement are genuine things to buy. Why did they not answer the question the engagement letter asked?

Because the clause was about disclosure to a third party, and sending records to a hosted service is a disclosure however good the terms are. A contract changes what the recipient may do with the material; it does not change the fact that the material was sent. It also does not move responsibility: Nikita is the one deciding to send it, and the agreement is evidence of due diligence rather than a transfer of liability. The two routes that actually answer the clause are consent from the client, or computation that never leaves the machine.

The machine would have cost four years of the subscription. Name two things that make that comparison worse than it looks.

First, the hardware only fits the model sizes it fits, and it will be superseded while the subscription's model improves under it at no extra cost. Second, running a model locally carries an ongoing cost nobody quotes: runners update and change behaviour, formats evolve, a driver stack breaks after an operating system upgrade, and better models arrive constantly, so the file installed today is quietly becoming the wrong choice. It is about an hour a month if kept up and a lost weekend if not.

MODULE 5 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 A model card says the weights are released under a community licence. What have you not necessarily been given?

A. The training data, and often the training code.

B. Permission to run it on your own hardware.

C. The right to quantise it for your own use.

D. A file format that any local runner on an ordinary laptop can load without conversion.

2 An eight-billion model runs comfortably on your sixteen-gigabyte laptop in chat, then crawls when you paste in a fifty-page report. Why?

A. The weights are loaded a second time in order to handle the longer input.

B. The working memory for the context grew until the machine began swapping to disk.

C. The model has exceeded its context window and is re-reading the document in passes.

D. Quantised models slow down in proportion to input length, which is the cost of the smaller file.

3 You have memory for either a large model crushed to two bits or a smaller one at four. Which is the better default?

A. The larger model at two bits, because parameter count dominates quality at any precision.

B. Either, because published perplexity figures show the two are equivalent at equal file size.

C. The smaller model at four bits, because degradation is not linear and two-bit is usually a false economy.

D. The larger model at two bits, provided you use an importance-matrix build, which restores the lost precision.

4 Why does this module tell you to prefer safetensors files over the older weight format?

A. The newer format is smaller, so the download costs less on a metered connection.

B. The older format cannot be quantised, which limits what your machine will run.

C. The newer format embeds the licence in the file, so you cannot use a model commercially without accepting terms.

D. The older format reconstructs objects by running code as it loads, so a hostile file can run commands.

5 Two providers serve the same open model name at different prices. What should you settle before comparing those prices?

A. The data terms, and the quantisation and context length each actually serves.

B. Which has more users, since a busier provider is better maintained.

C. Which one your router prefers by default, since that reflects measured quality.

D. Whether either has an outage history, because availability is what separates providers at this level.

6 The hybrid rule routes work by what the material contains rather than by how hard the task looks. Why?

A. Because small models handle hard reasoning as well as large ones do, which makes difficulty a useless axis.

B. Because a client name is a fact you can see, while difficulty is a judgement made under pressure.

C. Because local models run faster than hosted ones, so the hard work belongs on your own machine by default.

D. Because providers price by material type, so routing by content is also the cheapest arrangement available.

MODULE 6

Data, terms and the law

What happens to what you type is the question most likely to matter and least likely to be answered on a comparison page. This block reads a privacy policy in ten minutes, says what a business agreement actually buys and what no agreement can give you, sets out the shared shape of data protection law without pretending to give legal advice, separates the three tangled questions about who owns the output, and covers redaction, secrets, shared devices and the workplace policy your employer probably has not written.

BY THE END OF THIS MODULE YOU CAN

Establish for any tool whether your content is trained on, retained or read by people and whether that covers your tier, redact material so the sensitive part is never sent, recognise which questions about ownership and legality are genuinely unresolved, and write the one-page record of tools and boundaries that a client or regulator would ask for

48 · 8 min

Who reads what you type

THE ONE THING TO KEEP

Not training on your data and not storing it are different promises, and most providers only make the first.

Three questions, asked of every tool

1. Is my input used to train the model?
2. Is it stored, and for how long?
3. Can a human being read it?

These are separate questions with separate answers, and companies answer the first one loudly and the other two quietly. "We do not train on your data" is not "we do not keep your data." A provider can truthfully say the first while retaining your conversations for thirty days and allowing staff to review flagged ones.

The pattern, as of 2026

This is the shape it has held for a while. Verify it for your specific tool, because it changes.

Consumer free tiers. Frequently used for training by default. Usually there is a switch in settings, and it is usually not the first thing you see. In some jurisdictions the default is reversed by law.

Three separate promises, and only the first is made loudly

We do not train on your content	Commonly stated and easy to find. It says nothing about the two below it.
We do not store your content	A different promise. Thirty days for abuse monitoring is common, and deletion carve-outs for backups and legal obligations are close to universal.
No person will ever read it	Almost nobody promises this. Sampled and flagged conversations are reviewed by people, which is how abuse gets caught.

A provider can truthfully make the first while doing both of the others. Check which one the sentence you found actually says, and whether it covers your tier — free, paid consumer, business and API are frequently treated differently by the same company.

Consumer paid tiers. Varies by company, and this surprises people. Paying does not automatically opt you out. Several products have trained on paying subscribers' conversations, with a setting available to decline.

Business, team and enterprise tiers. Generally covered by a contract that says your content is not used for training. If you handle other people's information for a living, this is what you are actually buying.

API access. Across the major providers, API inputs are generally not used for training by default. Retention for abuse monitoring is common — often around thirty days — and zero-retention arrangements exist for customers who ask and qualify.

Local models. Nothing leaves your machine, provided the app around it is not sending telemetry. Check the settings once.

The sentence people miss

Somewhere in most privacy policies is a line saying that a sample of conversations, or conversations flagged by automated systems, may be reviewed by human beings for safety and quality.

This is not sinister — it is how abuse gets caught and how models get fixed. But it means the honest mental model is not "a machine reads this and forgets." It is "a machine reads this, and occasionally a person might."

How to check, in five minutes

- Open **Settings**. Look for **Data controls**, **Privacy**, or **Improve the model**. Read every toggle, including the ones already on.
- Open the privacy policy and use your browser's find function. Search for: *train*, *improve our models*, *human review*, *retention*, *delete*.
- For a paid work tool, find the trust or security page rather than the consumer policy. They often say different things.
- Take a screenshot with the date visible. Terms change, and a dated screenshot is the only record you will have of what you agreed to.

If you cannot find a clear answer in five minutes, treat that as the answer.

Rules that do not depend on trusting anyone

Do not paste what you could not email. Passwords, API keys, a customer database, unredacted medical or legal files. Not because a leak is likely, but because the cost of being wrong is far larger than the effort of not doing it.

Redact by role. The model does not need the name. "The patient," "the supplier," "the employee" produces the same quality of answer as "Mrs Adeyemi." This one habit removes most of the risk from most of the work.

Remember deletion has limits. You can usually delete a conversation. If data has already gone into a training run, it cannot be extracted from the trained model afterwards. Deletion going forward is real. Deletion going backwards is not.

Check whether it is even your decision. Your employer may have a policy. Your country may have a law — GDPR in Europe, the DPDP Act in India, LGPD in Brazil, POPIA in South Africa. Handling someone else's personal data usually shifts this from a preference to an obligation.

None of this requires paranoia. It requires knowing which of three promises a company is making, and reading the one sentence where they say it.

BEFORE YOU MOVE ON

Miguel upgrades to a paid consumer plan specifically so his conversations will not be used to train the model. Is that a safe assumption?

A. Yes. Payment creates a commercial relationship, and providers do not train on paying customers' content.
B. No. Some consumer plans train by default whether or not you pay; a settings toggle or a business tier is what changes it.
C. Yes, as long as he also turns off chat history, because history and training are the same control.
D. No, but it does not matter, because deleting a conversation removes it from the model.

49 · 8 min

Reading the terms in ten minutes

THE ONE THING TO KEEP

Search a privacy policy for train, retention, human review, third part, delete and transfer, read the sentence around each, check whether it covers your tier, and screenshot it with the date — and treat an unfindable answer as the answer.

You are not going to read all of it

A terms of service plus a privacy policy runs to twenty thousand words of careful drafting, and nobody reads them. The realistic goal is not to read them but to **find the six things that decide whether you can use this tool for the work you do**, which takes ten minutes with a browser's find function.

The search terms

Open the privacy policy and search for each of these. Read the sentence around each hit.

`train` or `improve our` — whether your content is used to develop the models. Note carefully whether the sentence covers your tier. Free and paid consumer tiers are often treated differently from business tiers and from the API.

`retention` or `retain` — how long content is kept after you send it, and after you delete it. Common answers are 30 days for abuse monitoring, or "as long as necessary", which means nothing you can plan around.

`human review` or `reviewers` — whether people may read samples or flagged content. Almost every service says yes somewhere. It is not sinister; it is how abuse is caught. It does mean the mental model of "only a machine sees this" is wrong.

`third part` — who else receives your data. Cloud hosts, analytics providers, moderation contractors, payment processors. A good policy lists its sub-processors on a linked page.

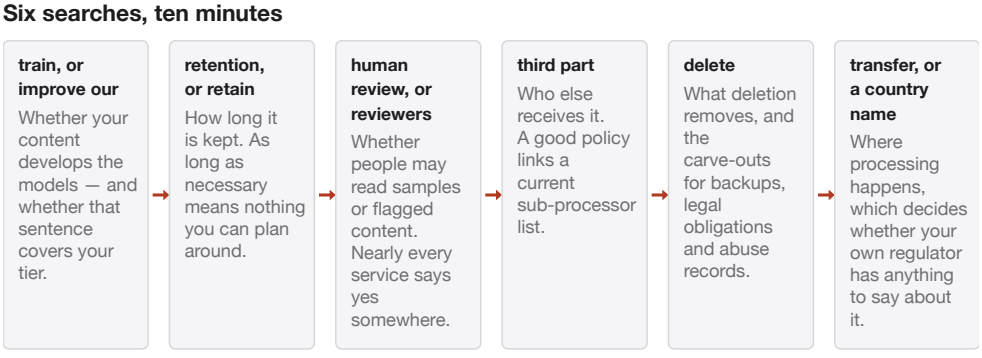
`delete` — what deletion actually removes, and what survives it. There is nearly always a carve-out for backups, legal obligations and abuse records.

`transfer` or a country name — where processing happens. This decides whether your regulator has anything to say about it.

The three pages that are not the privacy policy

The trust or security page. Usually where enterprise commitments live: certifications, sub-processor lists, regional processing options. Frequently says something different from the consumer policy, because it describes a different product.

The usage policy. What you are forbidden to do. Worth reading if your work sits anywhere near a restricted category — medical, legal, financial advice, or anything about elections. It binds you, and it is what an account gets suspended for.



Then screenshot the paragraph you relied on with the date visible. If you have told a client their material is not used for training, that screenshot is the only record you will have of what was true when you said it.

The data controls page inside the product. Settings, then Data controls or Privacy. Read every toggle, including the ones already switched on, and note which are on by default. This is where the actual switch is; the policy only describes it.

What "we do not train on your data" does not say

It is a narrow statement and it is worth reading precisely.

It does not say the data is not stored. It does not say no human will read it. It does not say it will not be sent to a third-party host. It does not say it cannot be produced in response to a legal demand. And it usually does not cover content you have flagged for feedback, or conversations reviewed after a safety trigger.

None of that makes the statement dishonest. It makes it specific, and the specificity is the point of reading it.

Where the answer differs by route

One thing worth checking rather than assuming: the same company frequently applies different terms to its chat product, its API and its business tier. The API is commonly the strictest — no training by default, short retention for abuse monitoring — while the consumer free tier is the loosest.

So "does this company train on my data" is not a well-formed question. The well-formed one names the route: *does this company train on what I send through the free web app, while signed into a personal account, with the default settings*. That is answerable, and the answer often differs from what a headline about the company said.

The screenshot

Take one, with the date visible, of the settings page and of the paragraph you relied on. Terms change. If you told a client that their material is not used for training, the only record you will have of what was true at the time is the screenshot you took.

For anything that matters, an archive service that captures a dated copy of a public page is better still, because it is not your own screenshot.

When the answer cannot be found

If ten minutes of searching does not produce a clear answer about training, retention and human review, treat that as the answer for anything confidential. Companies that have made a firm commitment say so plainly and near the top, because it is a selling point. Vagueness in this area is rarely accidental.

That rule alone will do more for your data safety than any amount of technical precaution, because it is applied before the material is sent rather than after.

BEFORE YOU MOVE ON

A provider states plainly that it does not train on customer data. What does that sentence leave open?

A. Nothing of consequence, since a commitment not to train implies the data is not retained for any other purpose.
B. Whether the data is stored, who may read it, where it is processed, and what a legal demand could compel.
C. Only whether the commitment applies retroactively to conversations from before the policy was published.
D. Whether the model has already memorised earlier inputs, which no current policy can undo once training has occurred.

50 · 8 min

What a business agreement actually buys

THE ONE THING TO KEEP

A business tier buys a contract rather than a better model — a training commitment, a data processing agreement, a sub-processor list, retention controls and regional processing — and none of it makes the output correct or moves the responsibility off you.

The tier difference is mostly contractual

Moving from a consumer plan to a business, team or enterprise plan rarely buys you a better model. It buys you a different contract, and for anyone handling other people's information that contract is the product.

Five things typically appear.

A commitment not to train on your content, as a term of the agreement rather than a setting you can toggle. The practical difference is that breaking a

contract has consequences that changing a default does not.

A data processing agreement. Under European and several other regimes, when you handle other people's personal data you are the *controller* and the service is your *processor*, and the law requires a written agreement between you setting out what the processor may do. Consumer terms are not that agreement. If you handle client data professionally, this document is the reason to upgrade.

A sub-processor list. Who else touches the data — cloud hosts, moderation vendors, analytics. Usually with a commitment to notify you before adding one.

Retention controls. Options ranging from a shorter default to zero retention for qualifying customers, meaning inputs are not stored after the response is returned.

Regional processing. A choice of where computation happens, which is what makes a residency requirement satisfiable.

Certifications, and what they mean

You will see SOC 2 Type II, ISO 27001, sometimes ISO 42001 for AI management systems, and sector schemes such as HIPAA arrangements in the United States.

Read them accurately. A certification says an auditor checked that the company follows the security processes it says it follows. It does not say the product is safe, that the model will not make things up, or that your data cannot leak. It is meaningful evidence about organisational discipline and nothing at all about model behaviour.

For most small organisations the certification matters mainly because a customer or an insurer asks for it. That is a legitimate reason to want it, and it is worth being clear that is what it is.

Reading a sub-processor list

The list is usually a linked page and it is more informative than the policy. It names every other company involved: the cloud provider running the hardware, any model provider if this one resells, moderation and support vendors, analytics, and payment processing.

Two things to look for. First, whether a model provider appears, because a product built on somebody else's model means your data reaches two companies and you should read both sets of terms. Second, where each sub-processor operates, since that is what determines whether data leaves your region. A provider whose page is current and specific is telling you something about how it is run, quite apart from what the list contains.

What no agreement gives you

It does not make the output correct. No contract addresses hallucination.

It does not remove your responsibility. If you feed a client's confidential material into a service, you are the one who decided to, and the agreement is your evidence of due diligence, not a transfer of liability.

It does not survive the model. A commitment about training is about the future. Anything already used in a completed training run cannot be extracted from the resulting weights.

It does not bind a reseller. If you reach the model through a wrapper, a router or an agency, your contract is with them and theirs may be different.

The realistic version for a small operation

Most readers are not negotiating enterprise agreements. The practical equivalent is short:

1. Use the API or a business tier where the default terms are better, rather than a consumer free tier.
2. Turn off training in settings wherever it exists, and screenshot it.

3. Redact by role so that the sensitive material largely does not arrive in the first place.
4. Keep genuinely confidential material on a local model.
5. Write one paragraph for your own records saying which tools you use, for what, and on what terms.

That last paragraph takes ten minutes and is what you will be asked for if a client, an employer or a regulator ever enquires. Almost nobody has one, and having one changes a difficult conversation into an easy one.

BEFORE YOU MOVE ON

An organisation upgrades to an enterprise plan and treats the SOC 2 report as evidence that the tool's answers can be relied on. What is the error?

A. An audit attests the company follows its stated security processes, not that answers are right.

B. The report describes the previous audit period, so it says nothing about the current version of the model.

C. SOC 2 covers security rather than privacy, so a separate privacy audit is required before the tool can be relied on.

D. SOC 2 applies to the provider's infrastructure and not to the customer's own configuration, which is where most failures occur.

51 · 9 min

Your country, your rules

THE ONE THING TO KEEP

Most data protection regimes share the same shape — you are the one deciding, you need a lawful basis and a purpose, people have rights, and sending data abroad is restricted — and your profession's rules sit on top of your country's.

Why this lesson cannot give you an answer

Data protection law differs by country, changes regularly, and applies differently depending on what you do and who your users are. Nothing here is legal advice, and anyone giving you a confident universal answer is guessing. What this lesson can do is show you the **shape** of the questions, so you know what to ask and whom to ask.

The pattern most of these laws share

Despite the differences, a common structure has spread widely — Europe's GDPR, India's DPDP Act, Brazil's LGPD, South Africa's POPIA, China's PIPL and others share most of these elements:

A distinction between the person who decides and the service that executes. In European terms, controller and processor. If you decide to send a customer's details to an AI service, you are the one deciding, and most obligations sit with you rather than with the provider.

A requirement for a lawful basis. You need a reason to process personal data, and "it was convenient" is not one. Consent, contract, and legitimate interests are the usual candidates, and which applies is a real question.

Purpose limitation. Data collected for one purpose cannot casually be used for another. Feeding your customer database into a model to see what it finds

is exactly the kind of thing this restricts.

Rights for the individual. Access, correction, deletion, and in some regimes objection to automated decision-making. That last one matters here: several regimes give people the right not to be subject to a purely automated decision with significant effects, which constrains using a model to decide about people without a human involved.

Rules on sending data abroad. Most of these laws restrict transferring personal data outside the country or region without a lawful mechanism. Since most AI processing happens in a handful of countries, this is where the practical friction lands.

Breach notification. A duty to report certain incidents within a stated time, often 72 hours.

Sector rules on top

Whatever your national law says, your profession may say more. Health, legal, financial, educational and government work carry additional duties, often with specific rules about disclosure to third parties, and those duties are usually owed regardless of what the tool's terms say. A clinician's obligation to a patient does not change because a service promised not to train on the note.

If you work in one of these fields, the question to ask is not "is this tool safe" but "does my professional body have guidance on this", and increasingly the answer is yes.

What actually changes what you do

Three practical consequences cover most situations:

Personal data means somebody's name, contact details, identifiers, or anything that could identify them. It is a wider category than people assume — a case reference plus a date of birth is personal data.

Where processing happens matters, and you can often choose. Regional endpoints, EU-hosted providers, and local models are the three answers, in ascending order of certainty.

The tool you choose is a decision you may have to justify. Not to prove it was the best choice, but to show you thought about it. A dated note saying what you considered is the whole of what "due diligence" means in practice for a small operation.

Automated decisions about people

One provision is worth singling out because it catches ordinary uses. Several regimes restrict decisions made about a person purely by automated means where the effect on them is significant — a rejected application, a refused loan, a disciplinary outcome, a candidate screened out.

The practical reading is that using a model to *help* you decide is different from letting it decide. Keeping a person in the loop who genuinely reviews and can overrule, and recording that they did, is what separates the two. If you are sorting job applications, assessing tenants or triaging benefit claims, this is the provision to look up for your own jurisdiction before you build the process, not after.

The honest limitation

This area is unsettled and moving. The EU's AI Act phases in obligations over several years; several countries are drafting AI-specific rules; enforcement priorities shift; and how existing data protection law applies to model training is being argued in courts and regulators' offices right now.

Anyone who tells you where the line is has told you where they think it is. Check the current position for your own country and your own profession, from a primary source, and treat everything written more than a year ago — including this — as a description of a moving thing.

BEFORE YOU MOVE ON

A small clinic wants to use an assistant on patient notes. Which question does the shared structure of these laws make most important?

A. Whether the provider is certified to a recognised security standard.

B. Whether the notes carry identifiers, what lawful basis covers sending them, and where it is processed.

C. Whether the model is accurate enough on clinical language to be safe for the intended use.

D. Whether patients have already given general consent to their notes being processed by any kind of computer system.

52 · 8 min

Who owns what it produces

THE ONE THING TO KEEP

Provider terms usually assign you their rights in the output, but whether that output is copyrightable at all and whether it infringes anyone else's rights are separate, unsettled questions answered differently in different countries.

Three separate questions people run together

1. Does the provider claim ownership of what you generate? Usually no. Most major providers' terms assign whatever rights they have in the output to the user, sometimes with conditions. That is a contract between you and them, and it is the easiest of the three questions.

2. Is the output protected by copyright at all? Much harder, and it differs by country.

3. Could the output infringe somebody else's rights? Different again, and the one with real exposure.

Conflating these produces most of the confusion in this area. A provider can perfectly well give you all their rights in something that nobody has any rights in.

Whether AI output is copyrightable

Copyright systems have generally been built around human authorship, and jurisdictions are reaching different conclusions about what that requires for machine-assisted work.

Some — the United States Copyright Office most prominently — have taken the position that material generated purely by a system in response to a prompt lacks the human authorship copyright requires, while a work containing sufficient human creative contribution can be protected as to that contribution. Some other jurisdictions, the United Kingdom among them, have long-standing provisions for computer-generated works with no human author, and the future of those provisions is under review. Others have not addressed it directly.

The consequences for you are practical rather than theoretical. If you produce a logo, an illustration or a piece of text entirely by prompting, you may not be able to stop somebody else using it. If that matters — a brand mark, a book cover, anything you intend to enforce — it is worth understanding your own jurisdiction's position before you rely on it, and worth documenting your own creative contribution.

Whether the output infringes

Two things are in dispute at once and it is worth separating them.

Whether training on copyrighted material without permission is lawful. Actively litigated in several countries, with different statutory starting points — some jurisdictions have text-and-data-mining exceptions, some do not — and no settled international answer. Cases are ongoing and settlements have been reached in some of them without establishing a general rule.

Whether a particular output infringes. Separate from the first, and more concrete. A model can reproduce training material closely, especially for

material that appeared very often. Generating something in the style of a named artist is generally a different matter from reproducing a recognisable work, and trademark law applies to logos and characters regardless of what copyright says.

Several providers now offer indemnities to business customers for copyright claims arising from output, usually with conditions such as using their safety filters and not deliberately prompting for protected material. If you are producing commercial work at scale, that clause is worth finding and reading.

What to do while it is unsettled

- **Do not generate a logo or a mark you intend to protect** without advice about your jurisdiction.
- **Do not prompt for a named living artist's style** for commercial work; it is the fact pattern most likely to attract a complaint, whatever the eventual law says.
- **Keep your prompts and your edits.** Evidence of human creative contribution is what several regimes turn on, and it costs nothing to keep.
- **Check whether your client's contract says anything.** Increasingly they do, and a clause requiring disclosure of AI use is now common in publishing, design and journalism.
- **Where the output must be clean, use tools with an indemnity** and follow its conditions.

Disclosure, which is a separate matter again

Whether you must tell somebody that AI was used is not a copyright question at all. It comes from contracts, from professional rules, and from platform policies — a publisher's submission terms, a client's agreement, a journal's authorship policy, an academic institution's rules, or a marketplace's requirement to label generated images.

These are proliferating and they vary widely, and breaching one is usually a contractual or disciplinary matter rather than a legal one. Check the specific

rules of wherever the work is going. The general legal position, whatever it turns out to be, will not help you if the client's contract said something else.

The honest position

This is genuinely unresolved, in different ways in different countries, and it will be resolved slowly and inconsistently. Anyone telling you that AI output definitely can or cannot be copyrighted, or that training definitely is or is not fair use, is describing a position rather than a settled rule. This lesson is not legal advice; where the answer carries money, it is a question for a lawyer in your jurisdiction, asked this year.

BEFORE YOU MOVE ON

A designer generates a logo entirely by prompting, and the provider's terms assign output rights to the user. Why might they still be unable to stop a competitor copying it?

A. Because the provider retains a licence to the output for service improvement, which weakens any exclusive claim.

B. A provider can only assign rights it has, and generated material may lack the human authorship required.

C. Because logos are protected by trademark rather than copyright, so the copyright question never arises for marks.

D. Because the same prompt could produce a near-identical image for the competitor, which defeats any claim to originality.

53 · 8 min

Redaction that actually works

THE ONE THING TO KEEP

Replacing identities with roles preserves everything the model needs and removes most of the risk, but files carry authorship metadata, hidden rows, extractable text under black rectangles and location data, and removing a name column is not anonymisation.

Replace the role, not the letters

The technique that does most of the work costs nothing and takes no time: write the role instead of the identity.

- "Mrs Adeyemi has missed three payments" becomes "the tenant has missed three payments".
- "Acme Ltd's contract with Northern Supplies" becomes "our client's contract with their supplier".
- "Patient DOB 12/03/1978, hypertension" becomes "a patient in their late forties with hypertension".

The answer you get back is the same. The model was never using the name for anything; it needs the relationship, the sequence and the numbers. Once this becomes a habit rather than a step, most of the confidentiality problem in most people's work disappears, because the sensitive material is not being sent at all.

When you need consistency

For longer documents where several people appear, replace consistently. Person A, Person B, Company X. Keep a small mapping in a file you do not send, and substitute back afterwards. This preserves the structure the model needs while removing the identities.

A find-and-replace in any editor does this in a minute. Free tools that automate it exist — **Microsoft Presidio** is an open-source library that detects and replaces names, numbers and identifiers in text, and spaCy's named-entity recognition does a rougher version in three lines of Python. For occasional work, doing it by hand is faster than installing anything.

The parts of a file you did not mean to send

This is where careful people still get caught, because the sensitive material is not in the text.

Document metadata. Word and PDF files carry author names, organisation, editing history and sometimes tracked changes and comments that are not visible in the document. Use the built-in inspector — *File, Info, Check for Issues, Inspect Document* in Word — before sending anything outward.

Spreadsheet hidden content. Hidden rows, hidden sheets, filtered-out data, and the source data behind a chart all travel with the file. Deleting the visible rows is not enough.

PDF layers. A black rectangle drawn over text hides it from the eye and not from a text extractor. This has produced published court filings where the redacted names could be copied out with a mouse. Proper redaction removes the text; the free **qpdf** and most PDF editors can flatten, and checking with `pdftotext` afterwards takes ten seconds.

Image metadata. Photographs carry EXIF data including GPS coordinates and the device. Free tools such as **ExifTool** strip it; most phones can be set not to record location in the first place.

Screenshots. The most common leak by far. A screenshot of one message includes the browser tabs, the notification that arrived, the file names in the sidebar and the account you are logged into. Crop before sending, always.

Why "anonymised" is harder than it sounds

Removing names does not always make data anonymous. A well-known line of research has shown that a surprisingly small number of ordinary attributes

— a postcode, a date of birth, a sex, or a handful of location points — is often enough to identify a specific person within a large population. Combining a de-identified dataset with a public one is a documented re-identification technique.

For a single document with one person in it, role substitution is genuinely effective. For a dataset of thousands of records, removing the name column is not anonymisation, and treating it as such is a mistake with regulatory consequences. If you are handling data at that scale, the question needs somebody who does this properly.

What redaction costs you

Be honest that it is not free. Removing detail sometimes removes information the model would have used: a name can carry a gender, a nationality or a professional register that changes how a letter should be written, and a company name lets a model apply what it knows about that industry.

Where that matters, put the useful attribute back without the identity — "a supplier in the garment trade", "a female patient in her sixties" — which supplies the context and not the person. And where the identity genuinely is the question, such as research about a named public figure, redaction is not applicable and the decision is simply whether to send it at all.

Hidden from the eye, or actually removed

Looks redacted	Actually removed
• A black rectangle drawn over text in a PDF	• Text deleted from the file, then checked with pdftotext
• Rows hidden rather than deleted in a spreadsheet	• A new file holding only the rows you meant to send
• A screenshot showing tabs, filenames and a notification	• Cropped before it leaves the screen
• A photograph with the location still in its metadata	• Metadata stripped with ExifTool, or never recorded
• A document carrying author name and tracked changes	• Inspected with the office suite's own document inspector

Role substitution does most of the work and costs nothing, because the model never needed the name — only the relationship, the sequence and the numbers. At the scale of a dataset rather than a document, removing the name column is not anonymisation and the question needs somebody who does it properly.

The rule that survives everything

Before pasting, ask one question: **could I email this to a stranger?**

If the honest answer is no, either redact it or use a local model. That single question, asked reliably, prevents more problems than any technical measure in this course, because it is applied at the only moment when prevention is still possible.

BEFORE YOU MOVE ON

A lawyer covers names in a PDF with black rectangles before uploading it. Why is this insufficient?

A. The rectangles are stored as a separate annotation layer that a viewer can toggle off with a keyboard shortcut.

B. Compression when the file is uploaded can degrade the rectangles enough for the text beneath to show through.

C. Drawing over text does not remove it, so the characters are still extractable from the file.

D. The rectangles remove the text from view but leave it in the document's metadata, where it can be recovered.

54 · 7 min

Secrets, and what not to paste

THE ONE THING TO KEEP

Never paste credentials or identity numbers at any tier, substitute placeholder values before copying config files and logs, and if it happens anyway rotate the secret first — deletion limits exposure but does not undo storage.

The short list

Some things should never go into a chat box with any provider, on any tier, under any terms, because the cost of being wrong is out of proportion to the convenience.

- **Passwords and passphrases.** Yours or anyone else's.
- **API keys, tokens and connection strings.** Including the ones sitting in a config file you paste to ask why the file is not working.
- **Private keys and certificates.**
- **Full payment card numbers, bank details, government identity numbers.**
- **Anything covered by a specific legal duty** you cannot discharge — sealed material, some categories of health and criminal record data, anything under a court order.
- **Whole databases of other people's personal information.**

The reason is not that a provider is likely to misuse them. It is that this material multiplies: it lands in conversation history, in backups, in an abuse-monitoring store, in a screenshot somebody takes, in an export you share with a colleague. Every copy is another chance, and none of them was necessary, because the model did not need the secret to answer the question.

The paste that catches developers

Debugging a configuration file or an error log is one of the genuinely best uses of these tools, and it is the most common way keys leak.

```
# what people paste
DATABASE_URL=postgres://admin:hunter2@db.internal:5432/customer
s
STRIPE_KEY=sk_live_51H...
Traceback: connection refused

# what the model needs
DATABASE_URL=postgres://USER:PASS@HOST:5432/DBNAME
Traceback: connection refused
```

The second version gets exactly the same answer. Make the substitution automatic: when you copy a config file or a log, replace the values before you leave the clipboard. It takes five seconds and it is the difference between a helpful conversation and a leaked credential.

Screenshots of a terminal are the same problem with none of the opportunity to edit. Terminal windows show environment variables, hostnames, usernames and paths that identify your employer. Crop, or paste text you have cleaned.

Code, and what it reveals

Source code carries more than logic: internal hostnames, table structures, business rules, customer names in test fixtures, and comments people wrote when they were annoyed. Before pasting a file from work, look at it as a document rather than as code.

If your employer has a policy, it governs. If your employer has no policy and you are pasting proprietary code into a consumer service, you are making a decision on their behalf that they may not have made.

After an accidental paste

It happens to careful people. The response, in order:

1. **Rotate the secret immediately.** A key, a password, a token — revoke and reissue. This is the only step that genuinely fixes anything, and it takes a minute.
2. **Delete the conversation.** Worth doing. It limits further exposure and it does not undo storage that has already happened.
3. **Check the retention terms.** You now know how long the copy persists.
4. **If it was somebody else's data, tell whoever is responsible.** Many organisations have a reporting duty with a clock on it, and the clock started when it happened rather than when you mentioned it.
5. **Write down what happened.** One paragraph. If it is ever asked about, the contemporaneous note is what makes it a handled incident rather than a concealed one.

Nobody has ever been in less trouble for reporting this quickly. The instinct to hope it does not matter is understandable and it is exactly backwards: the small immediate action is cheap, and the delay is what turns it into a serious matter.

Scanning before you share

If you regularly paste code or configuration, install a scanner once. **gitleaks** and **trufflehog** are free, run in a second, and detect the common key formats. Running one over a file before you copy it is a five-second habit that catches the credential you had forgotten was in there.

The same tools run as a pre-commit hook on a repository, which catches the other version of this mistake. Neither is specific to AI tools; both belong to any working life where secrets and text move around, and this is simply one more reason to have them.

The habit that prevents most of it

Keep secrets out of the things you paste from, rather than remembering to remove them at the moment of pasting. Environment variables instead of literals in config files, placeholder values in examples, and no credentials in test

fixtures. Then the careless paste, when it eventually happens, contains nothing worth having.

BEFORE YOU MOVE ON

Someone pastes a config file containing a live API key while debugging, then deletes the conversation. Why is the deletion not the important step?

A. Because deleted conversations remain visible in the account's export until the retention period expires.

B. The key may already sit in retention or backup systems; only rotating it removes the exposure.

C. Because deletion requests are processed asynchronously and may take up to thirty days to complete.

D. Because the model has already seen the key and could reproduce it in a later conversation.

55 · 7 min

Accounts, sharing and devices

THE ONE THING TO KEEP

Your conversations are visible to whoever holds the device, to an administrator on a work account, through browser sync and through any share link — so keep work and personal accounts apart, audit memory and connected apps, and treat a shared link as published.

Who else can see your conversations

Four routes, and most people have thought about none of them.

The person holding the device. A shared family phone or a household computer shows conversation history to whoever opens the app. For someone asking about health, money, immigration status, an abusive relationship or

their sexuality, this is a serious matter and it is invisible in every article about choosing AI tools. The answers are a private browsing window, deleting conversations, a device passcode, or a separate account the device does not stay signed into.

Your employer. If you use a work account, on a work device, or through your organisation's single sign-on, an administrator can typically see that you used it and often what you sent. That is normal corporate practice, not surveillance, and it is stated in the policy you accepted. It means work accounts are for work.

Browser sync. Signed into a browser that syncs across devices, your history and sometimes your open tabs appear on other machines you own — including the one at the office, or the family desktop.

Anyone you share a link with. Sharing a conversation usually creates a public or unlisted URL, and public conversation links have been indexed by search engines when a product made sharing more open than users expected. Assume a shared link is a published document.

Work and personal, kept apart

The rule is dull and it prevents a lot of trouble: **one account for work, one for personal, and never the same browser profile.**

The reason is not only privacy. Memory features build a picture from your conversations and apply it to later ones, so an assistant that has learned about your job will bring that context into a personal question and the other way round. Separate profiles keep the two apart cleanly.

If your employer provides a tool, use it for work. It is covered by their agreement, which is better than your consumer terms, and it puts the responsibility where it belongs.

Memory, and what it does with the past

Products increasingly remember across conversations — facts about you, preferences, ongoing projects. It is genuinely useful and it has three consequences

worth knowing:

It leaks between contexts. Something mentioned in one conversation appears in another, including one you are showing to somebody.

It is editable, and worth editing. There is a memory management page. Open it once; most people find something they would rather was not there, and something recorded wrongly that has been quietly shaping answers.

It is another copy of your information. Stored, retained, exported, and subject to the same policies as everything else.

If you are about to share your screen, or hand the device to a colleague, turn memory off or use a temporary chat. Most products have a mode that does not save.

Sharing an account, and why not to

Sharing a subscription is against most terms of service for simultaneous use, and the terms are the smaller problem. The real ones: everybody sees everybody's history, memory blends several people into one incoherent picture, a security event affects all of you, and there is no way to remove one person's access without changing the password for everyone.

Where cost is the reason — and it often is — the better answers are a team plan with separate seats, the API with a key per person, free tiers used deliberately, or a local model. All of them keep the histories apart, which is the part that actually matters.

Temporary chats, and what they do

Most products now offer a mode variously called temporary, incognito or "do not save". It keeps the conversation out of your history and usually out of memory, which is the right tool for a question you would rather not have appear on a shared screen later.

Read what it claims carefully, because the two halves differ. Not saving to your history is a visible, verifiable behaviour. Not retaining it at all on the provider's side is a separate promise, and several products still keep tempo-

rary conversations for a period for safety monitoring. It removes the risk from the person holding your device, which is the one it was designed for, and it is not the same as the material never having been sent.

The two-minute audit

Once, open the account page and check: which devices are signed in, which apps and integrations have access, what memory holds, whether any conversations are shared publicly, and whether the account has two-factor authentication. Remove what you do not recognise.

Most people doing this for the first time find at least one thing to revoke. It is the same audit you would do on any account holding years of your correspondence, which is exactly what this is.

BEFORE YOU MOVE ON

Why is a shared household subscription worse than a shared streaming account, beyond the terms of service?

A. Because usage caps are per account, so several people exhaust the strong model's allowance far faster.

B. Because the provider can detect simultaneous sessions and suspend the account without warning.

C. Because conversation histories cannot be deleted individually once several people have contributed to the same account.

D. Because everyone sees everyone's conversation history, and memory blends several people into one profile.

56 · 8 min

What your workplace probably has not decided

THE ONE THING TO KEEP

Check the confidentiality policy, your contract and your client agreements before asking about AI specifically, ask a specific bounded question rather than an open one, and keep a one-page record of which tools you use, for what, and what you never send.

The gap most people are standing in

A great many organisations have no AI policy, an out-of-date one, or a one-line ban that everybody ignores. Meanwhile the work is being done with these tools anyway. If you are the person doing it, you are making decisions on your employer's behalf, and the useful move is to make them deliberately and write them down.

This lesson is for the individual in that position, not for a compliance department.

Find out what already applies

Before asking anyone about AI specifically, check three documents you have already agreed to:

The confidentiality or IT acceptable-use policy. It almost certainly says something about not sending company information to third-party services. That clause covers AI tools whether or not it mentions them, and it is the one people breach without noticing.

Your contract. Clauses about intellectual property and about who owns what you produce may bear on work created with these tools.

Any client agreement you work under. Increasingly these carry a clause requiring disclosure of AI use, or prohibiting sending their material to third parties. In consulting, legal, publishing and design work this is now common.

That is fifteen minutes and it usually answers the question. Most "is this allowed" situations are settled by a policy that already exists.

Asking without triggering a ban

If you ask "may I use ChatGPT for work", a cautious answer is no, because no is the safe answer to an unbounded question. A specific one gets a better result:

> *"I want to use an assistant to draft first versions of supplier emails and to summarise public documents. Nothing with client names or personal data. Is that acceptable, or is there an approved tool you would prefer?"*

That is answerable. It names a use, states a boundary, and offers the organisation the option to direct you somewhere. It is also the paragraph you want on record, which is the real purpose.

Writing the one-pager

If you end up drafting something for a small team, the useful version is short and covers five things:

1. **Which tools are approved**, and which tier or account to use.
2. **What may never be sent** — client identifiers, personal data, credentials, anything under a confidentiality clause.
3. **What must be checked** before it goes out, and by whom. Anything client-facing, anything with a number in it, anything legal.
4. **Disclosure.** Whether clients are told, and in what circumstances.
5. **Who to tell when it goes wrong**, and that reporting quickly is expected rather than punished.

One page. A longer one will not be read, and this is a domain where a rule nobody remembers is worse than no rule, because it creates the impression of a

decision that has not been made.

The two failure modes

A blanket ban. People use the tools anyway, on personal accounts, on personal devices, with no oversight and worse terms. The organisation has traded a manageable risk for an invisible one. This is well documented and entirely predictable.

A blanket permission. Everybody sends everything to whichever free tool they found, and the first anyone hears about it is when a client asks.

The workable position is between them, and it is the one the one-pager describes: named tools, a clear line about what is sensitive, a check before anything goes out, and a route for reporting mistakes.

The check that has to be a person

One line in the one-pager does more work than the rest: what must be checked before it goes out. It works only if it names a person and a moment, rather than an intention.

"Client-facing text is read by the person sending it, before sending" is a rule. "We verify AI output" is not, and it is what most policies say. The distinction matters because the failures in this area are not people ignoring a rule; they are people believing somebody else already checked. Name who, and name when.

When you are the whole organisation

If you work for yourself, you are the policy. Write the paragraph anyway — which tools, for what, on what terms, what you never send — and keep it with your business records.

The reason is not bureaucratic. When a client asks whether you use AI on their material, and they increasingly do, the difference between a considered answer and an improvised one is the difference between reassuring them and worrying them. It takes ten minutes and you write it once.

BEFORE YOU MOVE ON

Why does a blanket ban on AI tools often leave an organisation worse off than a narrow permission?

A. The work happens anyway on personal accounts, replacing a managed risk with an invisible one.

B. Because competitors adopting the tools gain a productivity advantage the banning organisation cannot match.

C. Because a ban is unenforceable in law, so it exposes the organisation to claims from employees who breach it.

D. Because a ban prevents the organisation from negotiating an enterprise agreement with better data terms.

CASE STUDY · MODULE 6

Two hundred report cards and a parent's question

A school in Ranchi the week after term reports went home, trying to answer a parent who asked where her daughter's comments had been written

Class teachers at the school write four to six lines of comment for every pupil at the end of term, about forty pupils a class, nine classes in the middle school. It takes an evening a class and everybody dislikes it.

In September a teacher, Mr Barla, started pasting the term's marks and his rough notes into a free assistant and asking for a polished comment. The results were better than what he had been writing, and they were better still when he included the pupil's name, because the sentences read more naturally. By December most of the middle school's comments were being drafted this way. Nobody had decided anything. There had been no meeting.

At the parents' evening a mother who works as a software tester asked a straightforward question: which service was used, and what happens to what was typed into it.

Sister Mary Grace, the principal, did not know. The IT coordinator found out in ten minutes. A free consumer tier. A setting in the account, switched on by default, permitting the conversations to be used to improve the model. A retention statement that said content is kept as long as necessary, which is not a period anybody can plan around. And a paragraph, as almost every policy has, saying that samples and flagged conversations may be read by people.

The vice-principal wanted the practice stopped that evening. The position is defensible and it is what a parent would expect to hear.

Mr Barla's position was that the comments were better, that thirty-one teachers had just been handed back an evening a term, and that a ban would not return them to handwriting. It would move the practice to personal accounts on personal phones at home, where the school would know nothing about it at all.

Then a specific fact came out that changed the register of the conversation. One of Mr Barla's comments had explained a pupil's drop in attendance by referring to her father's illness, because that was the honest explanation and he wanted the comment to be fair. He had typed it, with her name, into a free consumer account, with no policy in front of him and no ill intent whatsoever.

Three things followed from that one comment and none of them was about the assistant. The information was about a third person who was not the school's pupil. It was health information, which in most data protection regimes sits in a more protected category than an address or a mark. And it had been typed into an account whose settings permitted the conversation to be used for training, by a teacher who had never been told there was a decision to make.

The IT coordinator raised a separate point that the meeting had not reached. Thirty-one teachers were doing this on whatever account they already had, which for most of them was the account they also use at home. Report comments, family circumstances, and the teacher's own private conversations were accumulating in the same history, on devices that in several cases are shared with children and spouses. Memory features in some of these products carry context from one conversation into the next, which means a teacher who has spent a term discussing pupils may find that context appearing in a personal question.

The vice-principal's answer to all of this was that it strengthened her case. The coordinator's view was that it strengthened the opposite one, because every fact she had just listed was a fact the school had learned by asking, and a ban would end the asking rather than the practice.

The cost on one side was a rule that somebody had to write, a route that somebody had to find, and a check that thirty-one teachers were following it, in December, in the last fortnight of a term. The cost on the other was a ban that looked responsible to a parents' meeting and would produce a worse, invisible version of the same practice by February.

What would you have written, and what would you have said to the parent?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

Neither the ban nor the shrug. Sister Mary Grace wrote a single page in an afternoon and it was in use before the next term began.

The approved account turned out to cost nothing: the school's existing productivity licence already included an assistant on business terms that nobody had switched on. The rule had five lines. No pupil names, no family circumstances and no medical information — roles only, so the pupil and a pupil whose attendance fell in October. Every comment is read by the class teacher before it is entered. Anything touching home circumstances is written by hand and typed nowhere. And the last line named a person to tell when something had already gone wrong, and said plainly that reporting it quickly was expected rather than punished.

Four teachers came forward in the first week. One of them was Mr Barla.

The parent received a written answer naming the service, the tier and the terms as they stood that week, with a dated screenshot kept in the school's file. She asked for it in writing, which is how the school came to have the paragraph it now gives to anybody who asks.

The comments written from roles rather than names read exactly as well as the ones written from names. The model had never been using the name for anything.

Why was it wrong to stop at whether the service trains on the school's data?

Because training, retention and human review are three separate promises with three separate answers, and companies answer the first loudly and the other two quietly. A provider can truthfully say it does not train on your data while keeping conversations for a period and permitting staff to read flagged ones. For a school

the retention and review answers are the ones that matter, because the material is a child's personal data and in one case her father's health, and the honest mental model is not that a machine reads it and forgets.

The vice-principal's ban was defensible. Why would it probably have left the school worse off?

Because the tools had already been adopted and the work had already got easier. A blanket ban does not return the practice to handwriting; it moves it to personal accounts on personal devices with worse terms and no visibility, which is the well-documented outcome. The school would have traded a risk it could manage for one it could not see, while believing it had made a decision. The workable position is named tools, a clear line about what is sensitive, a check before anything goes out, and a route for reporting mistakes.

What did the dated screenshot buy the school that the policy page itself did not?

Evidence of what was true when the school relied on it. Terms change without notice, and a parent or a regulator asking next year will be asking about the position this term. A dated capture of the settings page and of the paragraph relied on is the only record of what was agreed to at the time; the current live page will show whatever it shows then. For anything that matters, a dated copy held by an archive service is better still, because it is not the school's own screenshot.

MODULE 6 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 A provider states plainly that it does not train on your data. Which question does that statement leave open?

A. Whether you may use the service commercially under its terms.
B. Whether the model will produce accurate answers about your material.
C. Whether the content is stored, for how long, and whether a person may read a sample.
D. Whether the provider will erase your conversation history if you close the account and formally ask it to.

2 Why is 'does this company train on my data' not a well-formed question?

A. Because no company answers it honestly in a public policy document.
B. Because training is a technical process that privacy policies are not required to describe.
C. Because the answer differs between countries and no policy can state a position that holds everywhere.
D. Because the same company applies different terms to its chat product, its API and its business tier.

3 You upgrade to a business tier with a data processing agreement and a contractual no-training commitment. What has not changed?

A. That you decided to send the material, and you remain responsible for that decision.
B. Whether your content is used to develop the provider's models.
C. The retention period applied to your inputs.
D. Whether the provider must notify you before adding a new sub-processor to its published list.

4 Several regimes restrict decisions made about a person purely by automated means. What separates a permitted use from a restricted one?

A. Whether the model is hosted inside the country where the person lives.
B. Whether a person genuinely reviews the outcome and can overrule it, and that this is recorded.
C. Whether the person was told which model was used to assess them.
D. Whether the decision was reached by a model the regulator has formally certified for that category of use.

5 A PDF has a black rectangle drawn over a name before it is sent out. What is the state of the name?

A. Removed, because the rectangle is flattened into the page when the file is saved.
B. Removed from the page but retained in the document's metadata.
C. Still in the file, and recoverable by any text extractor.
D. Still in the file, but readable only by the software that created it, which limits the exposure considerably.

6 You realise you have pasted a live API key into a chat. What is the first thing to do?

A. Delete the conversation, which removes the copy from the provider's systems.
B. Check the retention terms to find out how long the copy will persist.
C. Export the conversation for your own records, then delete it and write a note of the incident.
D. Rotate the key: revoke it and issue a new one.

MODULE 7

Beyond the chat box

Most of the money in this field is spent outside the chat window, on tools nobody compares carefully. This block works through the categories: image generation, transcription and translation, voice, video, research and search, documents and OCR, spreadsheets and data, meeting notetakers, the per-seat office suite assistants, automation platforms, and the specialist products sold to your profession. Each one gets the same treatment — what genuinely separates the products, where the free path is good enough, and the specific way each category fails.

> BY THE END OF THIS MODULE YOU CAN
>
> Judge a tool in any of the main non-chat categories on the property that actually separates them — legible text, verifiable citations, licensed data, visible code, consent and retention — name the free path that does the same job, and identify the failure mode specific to that category before it costs you

57 · 9 min

Choosing an image tool

THE ONE THING TO KEEP

Image tools differ most in rendering text, keeping a subject consistent, and editing an existing picture rather than in raw quality — and the free local route removes both the per-image cost and the content restrictions, along with the indemnity.

Three kinds of product, priced very differently

Bundled inside a chat assistant. Convenient, included in a subscription you already have, weakest control. Fine for an illustration for a slide.

A dedicated hosted service. Midjourney, Adobe Firefly, Ideogram, Recraft and others. Better output, more control, a monthly fee or a credit balance. Each has a house style you will recognise once you have looked at a hundred images.

Open models on your own machine or rented hardware. Stable Diffusion and its successors, Flux, and others. Free after setup, unlimited, fully controllable, and the only route with no content restrictions — which is a benefit and a responsibility.

What separates them in practice

Text in the image. Historically the great weakness, now the clearest difference between products. Some render a shop sign or a poster headline correctly; some still produce plausible-looking nonsense. If your work involves any words in the picture, this alone decides the choice, and it is testable in one prompt.

Consistency across images. A character or a product that looks the same in six pictures. This is what separates a usable set from a collection of one-

offs, and the mechanisms differ — reference images, character features, or training a small adapter on your own pictures.

Editing an existing image. Inpainting a region, extending the canvas, removing an object, changing one element while keeping everything else. For commercial work this matters far more than generating from nothing, because most real jobs start from a photograph.

Style range. Some products push everything toward a glossy house look that is attractive and immediately recognisable. Others stay neutral and do what you ask. Neither is better; they suit different work.

The free path, properly

You can do all of this without paying anyone.

- **Fooocus** — the easiest way to run an open image model locally. Effectively one download, a simple interface, good defaults.
- **ComfyUI** — the powerful one. A node graph, a steep first hour, and complete control including inpainting, upscaling and multi-step workflows.
- **Krita with the AI diffusion plugin** — free painting software with generation built into a real editing tool, which is the most practical combination for anyone who actually retouches images.
- **GIMP** and **Photopea** for editing afterwards. Photopea runs in a browser, opens PSD files, and costs nothing.

A machine with 8 GB of video memory runs current open image models comfortably. Less than that works with smaller or quantised variants, more slowly.

The commercial question

Before using generated images in paid work, check three things.

The service's terms on commercial use. Most paid tiers permit it. Some free tiers do not, and some make your generations public by default, which is a problem for client work.

The model's licence, if you are running it yourself. Image models carry a wider range of licences than language models, including some that restrict commercial use.

Whether an indemnity is offered. Some providers train only on licensed or owned material and offer legal cover for business customers. If you are producing advertising or packaging, this is worth more than image quality.

And the unresolved part, stated as unresolved: whether training image models on scraped work was lawful is being litigated in several countries with different starting points, and whether a purely generated image can be copyrighted at all depends on jurisdiction. That affects what you can protect, not usually what you can make.

The two things that decide quality more than the model

A reference image. Nearly every capable tool now accepts one, and supplying a photograph of the product, the room or the person removes most of the guessing. A generated image built from a reference is far closer to what you wanted than the same tool given only words, and this is the single largest improvement available to a beginner.

Iterating rather than re-prompting. People generate, dislike it, rewrite the whole prompt and generate again. The productive habit is to keep the image you nearly liked and change one thing — inpaint the hand, extend the background, adjust the light — because you are refining a result rather than rolling dice a second time.

How to choose in one afternoon

Take three real jobs from your own work — not test prompts. Run each through two hosted services' free trials and one local setup. Judge on four things in this order: did it get the text right, could you fix the one thing that was wrong, does the style suit the client, and what will a hundred of these cost.

The last question is the one people skip, and it is why professionals who generate images all day end up running an open model locally regardless of which

hosted product they preferred.

BEFORE YOU MOVE ON

A designer producing packaging artwork with a client's brand name in the image should weigh which factors most heavily?

A. Style range and resolution, since packaging is judged on visual quality above all else.
B. Whether the tool renders text correctly, and whether commercial use and an indemnity are covered.
C. Speed and cost per image, since packaging work involves many iterations before approval.
D. Whether the model can be run locally, since client artwork should never be uploaded to a third-party service.

58 · 8 min

Transcription and translation

THE ONE THING TO KEEP

Transcription is available free and locally at commercial quality, so accuracy is decided by microphone placement, overlapping speech and vocabulary hints rather than by which service you pay — and both transcription and translation fail silently.

Transcription is a solved problem you can have for nothing

Whisper, released openly by OpenAI, transcribes dozens of languages at a standard that was commercial-grade when it appeared, and it runs on ordinary hardware. **whisper.cpp** makes it fast on a CPU; **faster-whisper** is quicker again on a modest GPU. There is a hosted version, and paid services

from several companies, and for most purposes the free local route is the correct answer.

The arithmetic that decides it: transcribing an hour of audio locally costs electricity and about five to fifteen minutes of your machine's time. The same hour through a paid service costs a small amount of money and about a minute. If you transcribe occasionally, either is fine. If you transcribe daily, local is free and private.

What actually affects accuracy

Not the service, mostly. In rough order of impact:

Audio quality. A clip-on microphone beats every software choice. Recording quietly, close to the speaker, in a room without hard echo does more than any model upgrade.

Overlapping speech. Two people talking at once defeats everything. This is the main reason meeting transcripts contain nonsense.

Domain vocabulary. Medical terms, product names, local place names, people's names. Most tools accept a hint list or prompt with expected terms, and supplying twenty words improves a transcript noticeably.

Accent and language. Performance is uneven and broadly tracks how much of that speech was in training. Test on your own recordings rather than trusting a published word error rate.

Speaker separation. Knowing *who* said what is a separate problem from *what* was said, called diarisation. It is markedly less reliable than transcription itself, and it is where paid services genuinely add value — **pyannote** is the main free option and it takes work to set up.

Translation: three different tools

Dedicated translation systems. DeepL and Google Translate. Fast, cheap or free, strong on common language pairs, and generally better than a general assistant at plain translation of ordinary prose.

General assistants. Better where context and judgement matter — a legal clause, a joke, a message that must land in a particular register, or anything where you can explain the situation. Worse at bulk.

Open translation models. NLLB-200 from Meta covers 200 languages; **MADLAD-400** from Google covers over 400. Both run locally and both matter enormously for languages the commercial services handle badly or not at all. **Argos Translate** packages offline translation for the desktop.

The workflow that beats each of them alone

For anything important: translate with a dedicated system first, then ask a general assistant to review the translation against the original, with the context explained. The second pass catches register errors, idioms taken literally, and terms that are technically correct and wrong in the setting.

For long documents, translate section by section. Quality degrades over long inputs, and a section boundary is a natural place to check.

Getting a transcript you can use

Two settings improve most transcripts more than any model choice. Ask for timestamps, so you can jump to the audio at the point where a number or a name appears and verify it in ten seconds rather than listening again from the start. And ask for the output as plain text rather than subtitle files if you intend to read it — subtitle formats break sentences at line lengths, which makes a transcript hard to read and harder to summarise.

```
whisper meeting.m4a --model medium --language en --output_for-
mat txt
```

The model size is the one real trade-off: larger models are more accurate and slower, and on a laptop the medium size is usually the sensible middle.

The limits worth stating plainly

Machine translation into a language you cannot read is a risk you are taking, not a task you have completed. For anything consequential — a contract, medical instructions, a safety notice — a human speaker needs to read it. This is not caution for its own sake: the failure mode is fluent, confident and undetectable from the side you understand.

Transcription errors are silent. A transcript reads perfectly while containing a wrong number, a wrong drug name or a negation dropped. Where the words matter, listen to the audio at the points that matter rather than trusting the text.

Recording other people has rules, and they differ by country and by setting. That is the subject of a later lesson in this module, and it applies before any of this does.

BEFORE YOU MOVE ON

A team's meeting transcripts are full of errors, and they are considering a more expensive transcription service. What should they change first?

A. The service, since paid providers are trained on meeting audio specifically and handle conversational speech better.

B. The recording setup, since overlapping speech and distant microphones cause more errors than model choice.

C. The language setting, since automatic detection often picks the wrong variant and degrades the whole transcript.

D. The post-processing, by passing the transcript through an assistant to correct errors against the meeting agenda.

59 · 8 min

Voice and speech tools

THE ONE THING TO KEEP

Text-to-speech is free, local and good enough for accessibility, proofreading and reaching people who do not read the language — while voice cloning carries a real fraud risk that makes explicit consent and a household callback rule the load-bearing parts.

What the category contains

Text to speech. Turning writing into audio. Free and built into every operating system; better voices available from paid services and from open models.

Voice cloning. Producing speech in a particular person's voice from a sample, sometimes only seconds long.

Speech to speech. Changing an existing recording to a different voice, keeping the timing and delivery.

Dubbing. Translating a recording and speaking it in the new language, sometimes matching the original speaker's voice.

The technology has moved fast, and the honest summary is that the good hosted services are now hard to distinguish from a person on a short clip, and the free options are good enough for most practical uses.

The free path

Piper runs locally, produces natural speech in many languages, and is fast enough on a Raspberry Pi. This is the workhorse for accessibility, for reading documents aloud, and for anything you produce regularly.

Coqui TTS and similar projects offer more voices and cloning at the cost of more setup.

Your operating system. Built-in voices have improved considerably. For reading a document back to yourself, no installation is required at all.

Where these genuinely help

- **Accessibility.** For someone who reads with difficulty or not at all, having any document read aloud in a natural voice is transformative, and it is free.
- **Proofreading.** Listening to your own writing exposes errors the eye slides over. This is an old editor's trick and it works better with a good voice.
- **Learning a language.** Hearing a sentence read correctly, on demand, at your own pace.
- **Reaching people who do not read your written language** but speak it — a large population in many countries, and one that written-only material excludes entirely.
- **Producing audio versions** of material at a cost that makes it worth doing for small audiences.

The consent problem, which is the important part

Voice cloning creates a genuine harm that has already happened many times. Recorded voices are used in fraud — the call that sounds like a family member in trouble, the instruction that sounds like a manager approving a payment. A few seconds of audio from a public video is enough for some tools.

Three things follow.

Do not clone a voice without the person's explicit permission. Reputable services require an attestation, some require a spoken consent recording, and it remains the user's responsibility. Several jurisdictions now have laws about voice likeness, and the rules differ.

Assume other people's voices can be cloned. The practical defence for your family and your workplace is a shared phrase or a callback rule for anything involving money or urgency. That advice sounds excessive until it has happened to someone you know.

Disclose synthetic voice where the listener would care. In advertising, in journalism, in anything a person might rely on. Some platforms and some regimes now require labelling, and the requirements are still forming.

Pronunciation, which is where most of the work goes

The gap between an acceptable read and a good one is almost entirely names, acronyms and foreign words. Every serious tool accepts a pronunciation override — a phonetic spelling, a dictionary of terms, or SSML markup that specifies how a word is said.

Building that dictionary once, for the twenty words your material keeps using, is an hour that improves everything you produce afterwards. It is also the reason a free local tool with a good dictionary often beats an expensive service used straight out of the box, and it is the part of the work no product page mentions.

Choosing a service

Beyond quality, three things separate them for real work:

- **Languages and accents**, particularly whether your language is supported at all rather than as an accented approximation.
- **Control over delivery** — pauses, emphasis, pace, pronunciation of names. On a long piece this is what turns an acceptable read into a good one.
- **Commercial terms.** Whether you may use the audio commercially, whether the voice is licensed for it, and what happens to the voice you cloned. Read this before recording a client's audiobook, not after.

For anything you produce regularly, test the free local option first. The gap between Piper and a paid service is real and it is smaller than the gap between doing this at all and not doing it.

BEFORE YOU MOVE ON

A company wants to use a cloned executive voice for internal announcements. Beyond audio quality, what is the most important consideration?

A. Whether the audio files are stored securely, since a leaked recording could be reused elsewhere.
B. Whether staff can tell the difference, since an announcement that sounds real but is not may reduce trust.
C. Whether the service supports the accents and pacing needed for the executive's usual delivery.
D. Explicit consent, and a verification rule so staff can tell a real instruction from a cloned one.

60 · 8 min

Video tools, generated and edited

THE ONE THING TO KEEP

Nearly all the practical value in AI video is in editing, where free tools like DaVinci Resolve are professional-grade, while generation is priced per second with a high discard rate and works best as short ingredients cut into real footage.

Two different products under one word

Video editing with AI assistance. Cutting, transcribing, captioning, removing filler words, reframing for a vertical screen, colour matching, separating a voice from background noise. This is where nearly all the practical value is, and it is mature.

Video generation. Producing footage from a text description or an image. Improving quickly, expensive per second, and useful for a narrower range of jobs than the demonstrations suggest.

Most people who say they want AI video want the first one.

The free path for editing is genuinely professional

DaVinci Resolve is free, used on released feature films, and includes AI-assisted transcription, subtitle generation, object removal, voice isolation and face refinement in the free version. The paid Studio version adds some features; the free one is not a trial.

Shotcut and **Kdenlive** are free and open source, lighter on hardware, and enough for most straightforward editing.

CapCut is free, phone-first, and good at the specific job of short social video with captions.

Against these, a subscription to a paid editor is buying a workflow, collaboration features, or a specific effect. That can be worth it. It is not buying capability the free tools lack.

What generation is and is not good for

Good for: short establishing shots, abstract or stylised sequences, background plates, product turnarounds, moving a still photograph slightly, and filling a gap where stock footage would otherwise be bought.

Bad for: anything with speech, anything needing a specific person to look the same across shots, precise action, hands doing detailed work, text on screen, and any sequence longer than a few seconds without a cut.

The mechanism explains the pattern. These models generate a short clip at a time, and consistency across clips is the hard problem. A one-minute finished piece is assembled from many short generations, most of them discarded.

Cost, which is the part people underestimate

Video generation is priced per second of output and the failure rate is high. A usable five-second shot commonly takes five to fifteen attempts. That makes the real cost of a shot several times the headline price, and a short film a serious budget.

Before committing to a project, generate one shot you actually need and count the attempts. The number is your multiplier for everything else, and it varies enormously by how specific your requirement is. Vague and atmospheric is cheap; a particular thing happening in a particular way is expensive.

Open video models exist and run locally on a strong GPU, which converts the cost into time and electricity. That is a real option for anyone doing this regularly with hardware to spare.

The workflow that works

Generated footage is best used as an ingredient. Generate the shots you cannot film, edit them together with material you have, and do the assembly in a real editor. Almost every good result you have seen was made this way rather than produced end to end by a single tool.

The same applies to the assistive features. Automatic captions are excellent and wrong often enough that you read them; automatic reframing is good and occasionally cuts somebody's head off; filler-word removal is fast and sometimes takes a breath the sentence needed. Each of these saves most of the work and none of them removes the review.

The assistive features worth having whatever else you use

Three of them justify installing a modern editor on their own, and all three are free in Resolve.

Automatic subtitles, which take a job that used to cost hours and make it a review pass. Subtitles also matter more than most people assume: a large share of social video is watched without sound.

Voice isolation, which separates speech from background noise well enough to rescue a recording made in a busy room. This is the feature most likely to save footage you thought was unusable.

Reframing for vertical and square formats, which tracks the subject and follows it. It gets it wrong often enough that you check every cut, and it still turns an afternoon into twenty minutes.

Disclosure

Requirements for labelling synthetic video are emerging in several jurisdictions and on most large platforms, and they differ. Platform rules are the ones you will meet first: most major video and social platforms now require creators to disclose realistic synthetic content, and enforcement is uneven but real. Check the rules of the place you are publishing, and be aware they change more often than the law does.

BEFORE YOU MOVE ON

A studio budgets a 60-second promotional film using generated footage at a quoted price per second. Why is the estimate likely to be badly wrong?

A. Generation prices are quoted at the lowest resolution, and production resolution costs several times more.

B. Sixty seconds exceeds the maximum length these models produce, so the film cannot be generated at all.

C. Generated footage requires colour grading and stabilisation in a paid editor, which the estimate omits.

D. Most attempts are discarded, so a usable shot often costs five to fifteen times its headline price.

61 · 9 min

Research and search tools

THE ONE THING TO KEEP

Search-grounded answers are bounded by what the search returned, so click two citations to check they support the claim and read the source list before the report — and where you can supply the sources yourself, the failure mode becomes a safe one.

Three products that look similar and are not

Search-grounded assistants. Perplexity and the search modes inside the major assistants. They run a search, read a few results, and answer with citations. Fast, good for current information, bounded by what the search returned.

Deep research modes. The assistant runs many searches over several minutes, reads dozens of pages, and produces a long structured report with sources. Genuinely useful for a survey of an unfamiliar area. Expensive in tokens and in waiting.

Grounded-on-your-own-documents tools. NotebookLM and similar. You supply the sources; the tool answers only from them, with citations back to the passage. This is a different and often better proposition, because the failure mode changes: instead of finding the wrong source, it tells you your sources do not say.

The failures, and how to catch them

The citation that does not support the claim. The most common failure and the easiest to miss. The link is real, the page exists, and it does not say what the answer says. Click two citations on anything you will rely on. Not all of them — two is enough to tell you whether this answer is the careful kind.

Two research tools that look alike and fail in opposite directions

Search-grounded answer	Grounded on documents you supplied
• Bounded by what the search engine returned	• Bounded by the material you chose
• Marketing pages rank well on commercial questions	• Citations point at a passage you can open
• A citation can be real and still not support the claim	• Questions outside the material get a refusal
• Paywalled, printed and other-language sources are invisible	• Retrieval can miss a passage worded differently
• Fails by finding the wrong source and sounding certain	• Fails by saying your sources are silent when they are not

Click two citations on anything you will act on — not all of them; two is enough to tell you whether this answer is the careful kind. Where you can supply the sources yourself, the failure mode becomes the safe one.

Search-engine junk as a source. The tool reads what the search engine ranked, and for commercial questions that is often marketing copy or an article written to rank rather than to inform. An answer built from three such pages is fluent and worthless.

Recency confusion. The model's training gives it an out-of-date belief; the search gives it a current page; the answer sometimes blends them. Watch for answers that mix a current fact with an obsolete one, particularly on prices, versions and job titles.

The one-source answer. Some answers rest entirely on a single page dressed up with several citations to the same domain. Check the source list for variety, not just for length.

Missing what is not on the web. Anything behind a paywall, in a book, in a database, or in a language the search did not query, is invisible. A survey of the literature built only from what is freely readable is a survey of what is freely readable.

What deep research is genuinely for

It is at its best when you know nothing about an area and need a map: who the main players are, what the standard positions are, what vocabulary to use, what to read next. As an orientation exercise it can save a day.

It is at its worst as a final answer on anything contested, because the length and structure create an impression of thoroughness that the underlying source selection may not deserve. Read the source list before the report. If the sources are weak, the report is weak however well it reads.

The grounded-on-your-own-documents option

For anyone with a body of material — a syllabus, a regulation, a set of contracts, a year of meeting notes — this is the most reliable configuration available to a non-programmer. The answers come from documents you chose, the citations point to a passage you can check in one click, and questions outside the material get "the sources do not cover this" rather than an invention.

It is not perfect. Retrieval can miss a passage that is relevant but worded differently, and an answer that says the sources are silent is sometimes wrong about that. But the direction of the failure is far safer than the alternative.

Choosing between them for a given question

The rule is short. If the answer must be current — a price, a rule, a fixture, who holds an office — use a search-grounded tool and click the citation. If the answer must come from a specific body of material you hold, use the grounded-on-your-own-documents route. If you need to understand an unfamiliar field, use deep research and treat its output as a reading list.

And if the answer is stable general knowledge, a plain assistant with no search is often better, because searching introduces the possibility of a bad page where the model's own knowledge was adequate. More retrieval is not uniformly safer.

The habit

For anything you will act on: read the sources, not the summary. The summary is a convenience for finding which sources to read, and treating it as the finished product is where people get caught. A researcher who uses these tools well spends the saved time on reading the three papers that matter rather than on the forty they no longer had to skim.

BEFORE YOU MOVE ON

A deep research report reads thoroughly and cites twenty sources, eighteen of them from one commercial domain. What does this indicate?

A. The topic is narrow enough that one authoritative source legitimately dominates the available material.
B. Deep research modes weight recent pages heavily, so an actively updated site will naturally dominate the sources.
C. The search tool failed to deduplicate results, so the same source was counted many times.
D. It rests on one perspective, and the citation count creates an impression of breadth it lacks.

62 · 8 min

Documents, PDFs and OCR

THE ONE THING TO KEEP

Check whether a PDF already has a text layer before anything else, because exact extraction beats recognition and both beat asking a vision model to read characters — and validate extracted tables by checking whether the rows still add to the printed total.

Ask one question before choosing anything

Does this PDF already contain text, or is it a picture of a page?

Everything follows from the answer, and it takes five seconds to find out. If you can select and copy text in a PDF viewer, the text is there. Or:

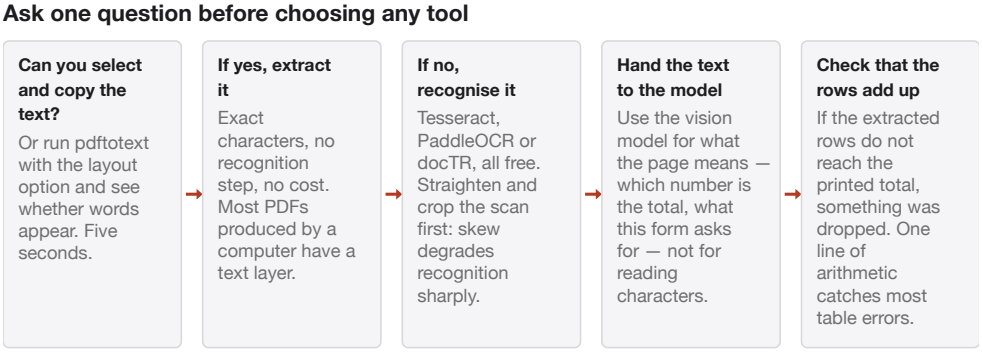
```
pdftotext -layout report.pdf - | head -20
```

If that prints words, the text layer exists and extraction is exact. If it prints nothing, the file is images and you need recognition, which is approximate.

People routinely send a text-layer PDF through a vision model, pay for it, wait for it, and receive a version with one digit changed. The text was sitting there.

The free tools, in order of what you should reach for

pdftotext (from Poppler) — exact extraction from PDFs with a text layer. `-layout` preserves column structure.



People routinely send a text-layer PDF through a vision model, pay for it, wait for it, and receive a version with one digit changed. The exact characters were sitting in the file the whole time.

Tesseract — the long-standing free OCR engine. Good on clean printed pages, weaker on complex layouts, supports over 100 languages including Indic and Arabic scripts.

PaddleOCR and **docTR** — free, more accurate than Tesseract on messy layouts, better on non-Latin scripts, more setup.

Docling and similar document-conversion tools — free libraries that turn PDFs into structured Markdown, preserving headings and tables, which is exactly the shape a model reads best.

LibreOffice — converts almost any office format to any other from the command line, free, and useful in a pipeline.

```
libreoffice --headless --convert-to pdf *.docx
tesseract scan.png out --psm 6 -l eng+hin
```

Where a vision model is the right tool

Not for exact characters. For **layout and meaning**: which of these is the invoice total, what does this form ask for, what is this diagram showing, is this table structured by row or by column. A vision model reading a page it can see, alongside text you extracted exactly, is a strong combination.

The general shape for a document pipeline: extract text with the exact tool, use the model to interpret, and check any number that matters against the source.

Tables, which are the hard part

Tables are where document extraction fails most often, in both routes. A table split across a page break, or with merged cells, or with a header repeated halfway down, confuses text extraction and vision models alike.

Two practical measures. Ask for the output in a structured form you can validate — a list of rows with named fields — so a missing column is visible rather than silently absorbed. And check the totals: if the extracted rows do not add up to the printed total, something was dropped, and this single check catches most table extraction errors in one line of arithmetic.

Scanned documents and forms

For a stack of scanned forms, the reliable approach is dull and works. Straighten and crop the images first — free tools do this, and a skewed scan degrades recognition sharply. Recognise the text. Then have a model pull named fields from the recognised text rather than from the image.

For handwriting, expect variable results and check everything. Neat printing is often fine; joined-up writing, faint pencil, and forms filled in by many different people are not. There is no tool that solves this reliably today, and a product claiming otherwise should be tested on your worst example rather than your best one.

Which format to hand the model

Once you have the text, the format you send it in changes the answers. Markdown with headings preserved is markedly better than a wall of undifferentiated text, because the structure tells the model what is a section, a heading and a list. This is why the document-conversion tools that emit Markdown are worth the extra step.

For long documents, keep the page or section numbers in the text as you extract, so that when you ask for a quotation the model can give you a location you can check. A citation you cannot verify is worth much less than one you can, and putting the numbers in costs nothing at extraction time and is impossible to add afterwards.

The one to remember

The cheapest, fastest and most accurate route through a document is almost always the least fashionable one: use the text that is already in the file. Reach for recognition when there is no text, and for a vision model when the question is about what the page means rather than what it says.

BEFORE YOU MOVE ON

A finance team extracts invoice tables with a vision model and some totals are wrong. What check would catch this reliably and cheaply?

A. Re-running each extraction twice and comparing, since inconsistent results indicate the model was uncertain.

B. Raising the image resolution before sending, so small print survives the model's downsampling.

C. Summing the extracted rows and comparing with the printed total on the document.

D. Asking the model to state its confidence for each figure it reads from the page.

63 · 8 min

Spreadsheets and data tools

THE ONE THING TO KEEP

Data work is the category where checking is cheapest, so choose tools that show you the code, and make printing row counts before and after every operation the habit that catches silently dropped records.

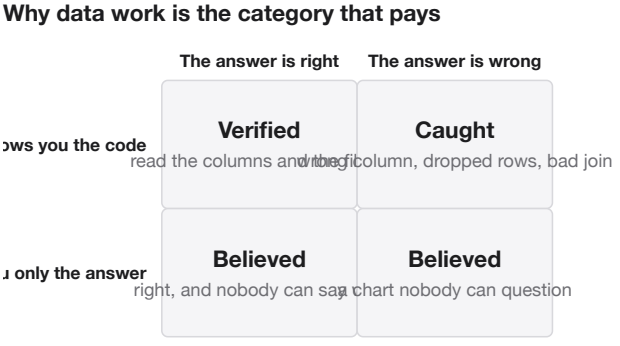
The category where verification is easy, and that changes everything

Most AI work is hard to check. Data work is the exception: if the model writes code, you can read the code, run it, and compare the answer with something you already know. That makes this the category where these tools give the most reliable return, and it should shape how you choose among them.

The consequence is a strong preference: **prefer a tool that shows you the code over one that shows you only the answer.**

Four shapes on offer

AI inside the spreadsheet. Formula suggestions, natural-language column operations, summarising a range. Convenient for small jobs, and it operates on your live sheet, which is worth respecting.



Which cell you land in is decided when you choose the tool, not when you read the output. Prefer the product that leaves a formula or a script over the one that pastes a value, even when the value is what you wanted.

Code interpreter in a chat product. Upload a file, ask a question, get an answer plus a chart plus the code that produced it. Currently the best general-purpose option for anybody who does not write code, precisely because the code is visible.

A notebook with an assistant. Jupyter or a hosted notebook with completion. More control, more setup, the same visibility.

SQL assistants. Turn a question into a query against a database. The query is the thing to read; a wrong join is invisible in the result and obvious in the SQL.

The free path

LibreOffice Calc for the spreadsheet. **Python with pandas** in a free notebook — Google Colab and Kaggle both offer free hosted notebooks, and a local Jupyter costs nothing. A free coding assistant such as Continue with a local model handles the writing. **DuckDB** runs SQL over CSV files directly, free, on your own machine, without a database server.

That combination does everything the paid data tools do for anyone working at the scale of a spreadsheet, and it keeps the data local.

The failures specific to data work

The silently dropped row. A filter or a join that removes records nobody counted. Always print the number of rows before and after every operation. This one habit catches more errors than any other.

The wrong column. Two columns with similar names, and the model picks the other one. Reading the code catches it; reading the chart does not.

Numbers that are text. A column of amounts stored as strings, with currency symbols or thousands separators, sorts alphabetically and sums to zero. Very common in exported data.

Dates. Ambiguous formats, mixed conventions in one column, and time zones. `03/04/2026` is two different days depending on where the file came from.

The confident average. A mean computed over a column containing sentinel values like `-1` or `9999` for "unknown". The answer is a number, and it is meaningless.

Silent type coercion. A join on an identifier where one side is numeric and the other is text produces zero matches, and the code runs without complaint.

The habit that makes it safe

Ask three questions of every result, and ask them out loud:

1. **How many rows went in, and how many came out?**
2. **Does the answer match something I already know?** One value you can verify by hand.
3. **What would this look like if the code were wrong?** If you cannot tell, add a check that would distinguish the two.

Then read the code. Even without knowing the language, the column names and the filter conditions are readable, and that is where the errors are.

Where the AI-in-the-spreadsheet products differ

If you do stay inside a spreadsheet, one distinction is worth knowing. Some products generate a **formula** you can inspect and keep; others compute a **value** and paste it in. The first leaves an auditable sheet that recalculates when the data changes; the second leaves a number nobody can explain in six months.

Prefer the formula every time, even when the value is what you wanted. A spreadsheet full of pasted constants is the same problem as an answer without code, and it is harder to spot because the sheet looks normal.

Where these tools are genuinely excellent

Exploratory work on a file you have never seen. Ask what is in it, what the distributions look like, what is missing, what looks odd. That is an hour of tedious work compressed into five minutes, it is exactly the kind of task where a wrong answer is visible, and it is the best argument for the whole category.

BEFORE YOU MOVE ON

An assistant joins two tables and returns a summary that looks plausible, but every row from one table is missing. Which check would have exposed this?

A. Printing row counts before and after the join and comparing with what was expected.

B. Asking the assistant to explain its approach in plain language before running it.

C. Comparing the summary against a figure you already know to be correct.

D. Requesting the output as a chart, since a missing group is visually obvious in a plot.

64 · 7 min

Meeting notetakers

THE ONE THING TO KEEP

Notetakers are adopted faster than the rules that govern them: consent for recording varies by jurisdiction, retention and who can search the archive are the questions to settle first, and summaries turn inconclusive discussions into decisions.

The most-adopted tool nobody evaluated

Automatic notetakers join a call, record it, transcribe it, and produce a summary with action items. They are genuinely useful and they are adopted faster than any policy about them, which is why this lesson is mostly about the parts that are not the product.

Recording other people has rules

They differ by country and by setting, and this is one of the few areas in this course where getting it wrong is straightforwardly unlawful rather than merely unwise.

Some jurisdictions require the consent of only one party to a conversation; others require the consent of everyone. Within federal countries this varies by state or province. In several regimes, recording people at work engages data protection law regardless of consent rules, and processing that recording — which is what transcription and summarisation are — is a separate matter from making it.

Three consequences that hold nearly everywhere:

Announce it. Most tools display a notice or a joined participant. Say it aloud as well. It costs three seconds and it is the difference between a courtesy and a complaint.

Let people decline. If someone objects, the recording stops. In practice this happens rarely, and how you handle it when it does is what people remember.

Do not record calls with people outside your organisation without asking first. Clients, candidates, patients, journalists. Ask before the call, not as it starts.

Where the participants are in different countries, the stricter rule is the safe one to follow.

What happens to the recording afterwards

This is the question people forget, and it is the one with the largest tail. Ask five things before adopting a tool:

1. **Where is the recording stored, and for how long?** Some retain indefinitely by default.
2. **Who in the organisation can see it?** Some tools make every meeting searchable by everyone, which is a surprise for the person whose one-to-one is in it.
3. **Is it used to train the provider's models?** Frequently yes on free tiers.
4. **Can it be deleted, and does that delete the transcript and summary too?**
5. **Does it integrate with anything else** — a CRM, a chat tool — that copies the content somewhere with different rules?

A tool that records performance conversations, salary discussions, medical consultations or legal advice needs the answers before the first meeting, not after.

Accuracy, and where summaries mislead

Transcription errors are covered earlier in this module; the summary adds its own.

Action items attributed to the wrong person. Common, and consequential, because people act on the list rather than the transcript.

Confident summaries of inconclusive discussions. A meeting that ended undecided is summarised as a decision, because summaries are written in the shape of conclusions.

Nuance removed. "We could do X, though Y worries me" becomes "agreed to do X".

The practical defence: the summary is a draft, the person who ran the meeting reads it before it circulates, and anything with a name and a deadline on it is checked against what was actually said.

The free path

Record locally, transcribe with Whisper on your own machine, and summarise with whatever assistant you already use. It takes ten minutes to set up, costs nothing, and keeps the recording out of a third party's storage entirely — which for sensitive meetings is not a saving but a requirement.

What to check in the first week

Run the tool alongside your own notes for three meetings before relying on it. Compare on two things only: did it capture every decision, and did it attribute the actions correctly. Everything else about a notetaker is convenience; those two are what people act on.

Watch particularly for meetings where somebody dialled in on a poor line, or where two people speak over each other regularly. Those are the meetings where the transcript degrades most, and they tend to be the same meetings every week, which means the failure is predictable and can be planned around rather than discovered.

The question worth asking first

Would this meeting be different if everyone knew it was on the record? If the answer is yes, that is not an argument for hiding the recorder. It is informa-

tion about whether to record at all.

BEFORE YOU MOVE ON

A manager records one-to-ones with a notetaker whose archive is searchable across the company. What is the most serious problem?

A. Sensitive conversations are readable by everyone in the company, which the participants did not agree to.
B. Summaries of one-to-ones tend to overstate agreement, which could misrepresent a performance discussion.
C. Retention policies rarely cover one-to-one meetings explicitly, so the recordings may be kept indefinitely.
D. Transcription errors in personal discussions are harder to correct than in project meetings.

65 · 8 min

Office suite assistants

THE ONE THING TO KEEP

An office suite assistant sells access to your own mail and files inside the organisation's boundary rather than a better model, so pilot five seats across different roles and read the distribution — and tidy file permissions before deploying, not after.

The per-seat question

Microsoft and Google both sell an AI assistant embedded across their office suites, at a per-user monthly price that has generally sat in the region of a consumer assistant subscription or above, on top of the licence you already pay. For an organisation of any size that is a substantial number, and it is usually decided by whoever is most enthusiastic rather than by measurement.

The decision deserves the same treatment as any other in this course: what does it do that a $20 consumer subscription plus copy and paste does not?

What the integration genuinely buys

Access to your own material without uploading it. The assistant can reach your mail, files, calendar and chat history within the tenant. Asking "summarise the thread with the supplier about the delayed shipment" works because it can find the thread. That is a real capability and it is the main thing you are paying for.

It stays inside the organisation's boundary. Content does not leave for a third-party service, which is often what makes the tool permissible at all in a regulated setting. For many buyers this, not capability, is the actual reason.

Meeting summaries and recaps tied to the calendar and the recording, with attendance and follow-ups.

In-place editing. Rewriting inside the document rather than in a chat window and back. Small, and it adds up over a day.

What it does not buy

A better model. These products run the same class of model as the consumer assistants, sometimes an older or smaller one for cost reasons.

Reliability on your data. Retrieval across a large organisation's files is genuinely hard, and the common complaint is not a wrong answer but an unhelpful one — it did not find the document, or it found an old version. Test on your own messy shared drive, not on a demonstration tenant.

Freedom from checking. A summary of a mail thread can drop the qualifying sentence, and a document draft can invent a figure. The material is yours, which makes errors more plausible and therefore easier to miss.

The evaluation to run before buying seats for everyone

Buy five licences for a month and pick five people who do different jobs. Ask each to keep the two-week line-a-day record from earlier in this course: min-

utes saved, minutes spent checking, and one thing it could not do.

Then look at the distribution rather than the average. The usual finding is that it is transformative for two roles — often anyone who lives in a mailbox or writes repetitive documents — and close to useless for the rest. That is an argument for buying it for those roles, which is a very different purchase from buying it for everybody.

Permissions, which is where this bites

An assistant that can search everything the user can search will surface things the user could always technically reach and never actually found. Over-broad file sharing that was harmless when nobody browsed the shared drive becomes visible when a natural-language question retrieves it.

The right sequence is to tidy permissions first and deploy second. Organisations that do it the other way round discover their sharing practices through an assistant cheerfully quoting a salary spreadsheet that was shared with everyone in 2019.

The licensing trap

One detail catches organisations out at renewal. These assistants are usually sold as an add-on to a specific licence tier, and buying them sometimes requires upgrading the underlying licence first. The advertised per-seat price is then not the increase you experience.

Ask for the total change to the invoice rather than the price of the add-on, and ask whether the commitment is monthly or annual. Annual commitments are common here and they lock in a decision about a fast-moving product for a year, which is exactly the situation where a pilot on five seats first is worth the delay.

The free path

For an individual or a small team: a consumer assistant subscription, plus the browser extensions or the desktop app for context, plus copy and paste. You

lose the automatic access to your own material, and you keep most of the benefit for a fraction of the price.

For a small organisation, that arrangement plus a clear rule about what may be pasted is frequently the correct answer, and the per-seat product becomes worth it at the point where searching your own material is the bottleneck rather than the writing.

BEFORE YOU MOVE ON

An organisation deploys a suite assistant and staff begin surfacing documents they were never meant to browse. What went wrong?

A. The assistant's index ignored folder permissions, which is a configuration error in the deployment.

B. Over-broad sharing became findable, because search reaches material nobody used to browse.

C. Retrieval surfaced cached older versions of documents whose permissions had since been tightened.

D. Staff were given administrative roles during the rollout to simplify configuration, expanding what they could see.

66 · 8 min

Agents and automation platforms

THE ONE THING TO KEEP

Most real value is in fixed workflows with a model in one step rather than in self-directing loops, because per-step reliability compounds badly — and any agent that reads untrusted text while holding permission to act is an unsolved security problem.

What the word means on a product page

"Agent" now labels at least four different things, and knowing which one you are buying prevents most disappointment.

A chat assistant with tools. It can search, run code, read your files. Useful, well understood, and not autonomous in any interesting sense.

A workflow with a model in it. A defined sequence — new email arrives, extract fields, write to a spreadsheet, send a reply — where a model does one or two steps. Reliable, because the sequence is fixed and only the judgement is delegated.

A loop that decides its own steps. Given a goal, it plans, acts, observes and repeats until it thinks it is done. This is what the research community means, and it is the least reliable of the four.

A hosted product with a chat box. Marketing.

Most business value today is in the second category, and most of the excitement is about the third.

Why the autonomous loop is hard

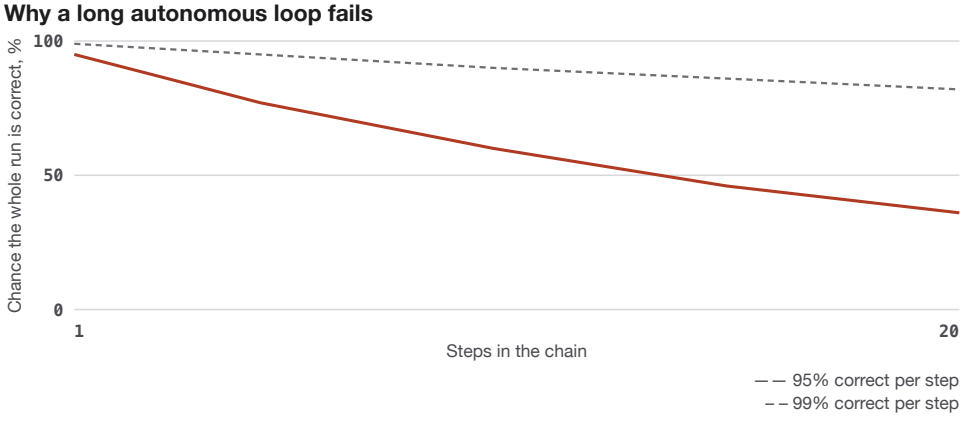
Reliability compounds badly. If each step is right 95% of the time, ten steps in sequence succeed about 60% of the time; twenty steps, about 36%. Errors do

not merely accumulate, they mislead — a wrong observation at step four sends the plan somewhere useless for the next six.

This is not an argument against agents. It is an argument for **short loops with checkpoints**: three or four steps, then a result a person or a rule can verify, then the next three. Every reliable system built this way looks less impressive in a demonstration and works far better on a Tuesday.

The no-code platforms

Zapier and **Make** connect hundreds of services with a model step available. Fast to build, priced per task run, and the price grows with volume in a way that surprises people at the second invoice.



Errors do not merely accumulate. A wrong observation at step four sends the plan somewhere useless for the next six. The answer is short loops with checkpoints: three or four steps, then a result a person or a rule can verify, then the next three.

n8n is the free, self-hostable equivalent. More setup, no per-run cost, and your data stays where you put it. For anything running at volume, or touching material you would rather not route through a third party, this is the one to look at.

Power Automate and the equivalents inside office suites, which are already licensed in many organisations and often overlooked.

Building the same workflow in a scripting language is also free, more flexible, and harder to hand over to a colleague. The platforms earn their place mainly

by being maintainable by somebody other than the person who built them.

The questions to ask before switching one on

1. **What can it do that cannot be undone?** Send mail, post publicly, move money, delete records. Those steps need confirmation, always.
2. **What does it do when it fails?** Stop and alert, or retry silently forever?
3. **Is there a rate limit on its own actions?** An agent in a loop can send four hundred emails.
4. **Who sees the log?** You need to be able to reconstruct what it did and when.
5. **What does it read?** If it processes incoming mail, web pages or tickets, that content can contain instructions aimed at it. An agent reading attacker-supplied text with permission to act is the central unsolved security problem in this area, and it has no complete fix today — the practical mitigations are least privilege, confirmation on consequential actions, and not letting one agent both read untrusted input and act on sensitive systems.

Cost, which behaves differently here

Automation platforms charge per task run, and a workflow that seemed cheap in testing changes character at volume. A run that costs a fraction of a penny is nothing at fifty a day and a real bill at fifty thousand a month, and the model call inside it is charged separately on top.

Work out the monthly cost at your expected volume before building, then double it, because workflows acquire steps. Self-hosting removes the per-run charge entirely and replaces it with a server to look after, which is the right trade above a few thousand runs a month and the wrong one below it.

The sensible starting point

Automate something small, boring and reversible. Sorting incoming messages into folders. Drafting — not sending — a reply. Extracting fields from attachments into a spreadsheet. Watch it for a fortnight before extending it.

The failure that costs people is the ambitious first project: a system that touches customers, money or public output before anybody has learned how it behaves when it is wrong. It will be wrong, on a schedule you did not choose, and the only question that matters is what it is holding at the time.

BEFORE YOU MOVE ON

A ten-step autonomous agent whose individual steps are right 95% of the time completes its task correctly about 60% of the time. What follows for the design?

A. Use a more capable model, since raising per-step accuracy to 99% would bring the ten-step success rate above 90%.

B. Add retries at each step, since a failed step retried three times succeeds almost always.

C. Accept the rate, since 60% autonomous completion still saves more time than doing all ten steps by hand.

D. Break it into short runs with a verifiable checkpoint after every three or four steps.

---

67 · 8 min

Specialist and industry tools

THE ONE THING TO KEEP

A specialist tool is worth its premium when it has licensed data you cannot get, cites into verifiable sources, integrates with your working system, or carries accountability — and the professional stays responsible for the advice regardless.

What a vertical product adds, and what it does not

Every profession now has AI products aimed at it: legal research and contract review, clinical documentation, accounting and audit, recruitment, property,

insurance, education. They cost several times a general assistant and sometimes considerably more.

The honest position is that some are worth it and many are a general model with a prompt and an industry logo. The difference is checkable, and it comes down to four things.

Does it have data you cannot get? A legal tool with a licensed case database, a medical tool with a drug interaction dataset, an accounting tool connected to current tax tables. This is the strongest form of added value, because no prompt substitutes for it.

Does it cite into a source you can verify? In professional work this is the difference between a usable tool and an unusable one. An answer with a link to the paragraph is checkable; an answer without one has to be re-derived, and you have saved nothing.

Is it integrated with the system you work in? The practice management system, the electronic health record, the ledger. Integration is genuinely expensive to build and genuinely valuable.

Does anyone stand behind it? Professional indemnity, an accuracy commitment, a support arrangement with somebody accountable. Most general tools explicitly disclaim all of this.

If the answer to all four is no, you are looking at a wrapper, and the useful comparison is against a general assistant with a well-written prompt.

The evaluation that settles it

Take five real matters from your own work where you already know the answer, and where you remember the subtlety that made each one difficult. Run them through the specialist tool and through a general assistant.

Then score three things: was it right, did it cite something you could check in under a minute, and did it notice the subtlety. That third column is where the specialist products either justify themselves or do not, and it is the one no demonstration will show you, because demonstrations use straightforward cases.

The questions a demonstration will not answer

Three things to ask a salesperson, in these words.

"Which model is underneath, and what happens when it is deprecated?" A product built on somebody else's model inherits that model's retirement schedule, and a wrapper that cannot answer this has not thought about it.

"What happens when it is wrong, and how would I know?" The good answer describes a citation you can check or a confidence signal you can act on. The bad answer is that it is rarely wrong.

"What is in your training or reference data, and when was it last updated?" For anything governed by rules that change — tax, regulation, clinical guidance — a reference set updated annually is a different product from one updated weekly, and the difference is invisible in a demonstration.

Where the responsibility sits

This matters more in professional work than anywhere else in the course, and it is simply stated: **the professional remains responsible for the advice.**

Courts have sanctioned lawyers for filed citations that did not exist. Regulators have acted on clinical documentation errors. In every case so far the answer has been that the tool is not the professional, and using one does not transfer the duty of care.

Two things follow. Read your professional body's guidance, because most now have some, and it is usually more permissive and more specific than the rumours circulating in your field. And where you use one of these tools on a client matter, know whether your engagement terms and your insurer require you to say so.

The free comparison worth running

Before buying anything in this category, spend an hour building the general-assistant version of it: a well-written prompt describing your jurisdiction, your

standards, your format, and the checks you always apply, saved as a project or a custom instruction. Attach your own reference documents where the tool allows it.

That takes an hour and costs nothing. Sometimes it is 80% of the specialist product, and knowing that changes what you are willing to pay for the remaining 20%. Sometimes it is visibly worse, and then you have learned that the specialist tool is doing something real — which is exactly what you wanted to find out.

BEFORE YOU MOVE ON

A legal research tool costs ten times a general assistant. Which single property most justifies the difference?

A. It carries a licensed case database and cites into passages you can verify in under a minute.

B. It has been fine-tuned on legal language, so it produces answers in the correct professional register.

C. It refuses questions outside its domain, so it cannot stray into areas where it would be unreliable.

D. It is marketed to the profession and therefore designed around the workflows lawyers actually use.

CASE STUDY · MODULE 7

Twelve thousand licence applications and a question nobody asked

A municipal corporation office in Jaipur evaluating a quotation to digitise a decade of trade licence applications

The office holds twelve thousand four hundred trade licence applications from 2016 onwards, about fifteen thousand eight hundred pages. They are needed as searchable records with eight fields each: licence number, applicant name, trade category, ward, date of application, fee paid, date of issue and renewal status.

A vendor quoted fourteen rupees a page for what the proposal called AI-powered document intelligence. That is about two lakh twenty thousand rupees, six weeks, and a contract with somebody accountable at the end of it. The demonstration used three clean forms from 2023 and read all eight fields correctly from each.

A junior assistant, Rakesh Meena, asked one question in the meeting. Are these scans, or are some of them files the corporation's own licensing software printed?

Nobody knew, so they checked. Four thousand nine hundred of the files had been generated by the office's own system after 2021 and carried a text layer. A free command-line tool read every character of them exactly, in seconds, with no recognition involved and no possibility of a digit being misread. The remaining seven thousand five hundred were genuine scans of paper forms — many completed by hand, some skewed, a number of them photocopies of photocopies.

So the quotation covered four thousand nine hundred files for which the answer was already sitting in the file.

The decision that followed had a cost on both sides. The vendor's price covered everything, arrived on a date, and came with somebody to complain to. Doing it in-house with free tools meant Rakesh and one colleague spending

about five weeks of their own time in an office that is short-staffed, and if it went wrong there would be nobody to complain to but themselves.

Rakesh's supervisor, Mrs Sharma, raised the point that settled the shape of the work. Two of the eight fields are numbers. A wrong trade category is an annoyance that somebody will notice and correct. A wrong fee in a licence record is a discrepancy against the treasury's books that may not surface for years.

And nobody had tested the vendor's tool on the worst material. Handwritten applicant names in Devanagari on a faint photocopy are a different problem from a clean printed form from 2023, and the demonstration had not contained one. When Mrs Sharma asked for a demonstration on twenty files the office would choose, the vendor's representative agreed readily, which was itself informative, and then asked for a week.

Rakesh spent an evening of his own on the four thousand nine hundred text-layer files to find out whether the in-house route was fantasy. It was not. Exact extraction took minutes. The part that was not trivial was the fields: the forms changed layout twice between 2016 and 2025, the ward is written in three different formats, and about a fifth of the applications record the fee in words as well as figures, which disagree in a handful of cases.

That last discovery mattered more than the vendor's price. It meant that whichever route the office took, somebody was going to have to decide what happens when a form says one thing in figures and another in words, and no tool was going to decide it for them.

Mrs Sharma's own position was that the office had been offered a service and had not yet worked out what it was buying. Two lakh twenty thousand rupees for pages, or a database the treasury would accept? Those are different products and only one of them was in the quotation.

What would you have quoted for, and what would you have written into the contract?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

They split the job, which neither side had proposed.

The four thousand nine hundred text-layer files were extracted in-house in two days. Exact extraction, then a model asked to pull the eight named fields from the extracted text rather than from a picture of the page. Total cost in usage: about nine hundred rupees.

The seven thousand five hundred scans went to the vendor, re-quoted at the same rate for nine thousand one hundred pages: one lakh twenty-seven thousand instead of two lakh twenty.

Two conditions went into the contract. The vendor returns the recognised text alongside the extracted fields, so any disputed record can be traced back to what was actually read. And every batch is checked by a rule the office wrote itself: the fee column is summed by month and compared against the treasury's own monthly receipts. The first batch was out by forty-one thousand rupees in one month — sixty-one records in which a handwritten five had been read as a six on fees ending in five hundred. That batch was re-run.

Handwritten names stayed the weak part. The office accepted it deliberately, because every record also carries a printed licence number, so a wrong name is recoverable from the number. That was written down as a decision rather than discovered later as a fault.

One question moved the quotation by nearly a lakh. What was it, and why does it come before every other tool choice for documents?

Whether the PDF already contains a text layer or is a picture of a page. If the text is there, extraction is exact and free, and takes seconds. If it is not, you need recognition, which is approximate and can change a digit without saying so. Sending a

text-layer file through a recognition or vision route is strictly worse in every dimension — slower, more expensive, and capable of introducing an error where the exact characters were already sitting in the file. It takes five seconds to check.

The monthly fee check caught sixty-one errors in one batch. Why is that check worth more than a better recognition engine?

Because extraction errors are silent. A record with a wrong digit looks exactly like a correct one, and no amount of engine quality removes the possibility on faint handwriting. A validation that compares extracted totals against a figure known independently turns a silent error into a visible one, in a single line of arithmetic, on every batch. It is the document equivalent of printing row counts before and after an operation, and it catches the errors that reading the output never would.

The office accepted a known weakness on handwritten names. What made that acceptable here, and when would it not be?

It was acceptable because each record carries an independently printed licence number, so a misread name is recoverable rather than lost, and because the limit was recorded as a decision with a reason rather than left to be discovered as a failure. It would not be acceptable where the name is the only key, where a wrong name has a consequence that cannot be reversed, or where nobody has written down that the weakness exists — at which point the next person to use the data will reasonably assume it is sound.

MODULE 7 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 Before choosing any tool for a PDF, this module says to establish one thing first. What is it?

A. Whether the file has a text layer, because extraction is exact and recognition is approximate.

B. Whether the file is under the tool's upload limit, because larger files are silently truncated.

C. Which language the document is in, because OCR engines must be told before they will read a page.

D. Whether the pages are the right way up, because a rotated page defeats extraction and recognition equally.

2 Which habit catches more errors in data work than any other?

A. Asking the model to state its confidence in the result.

B. Printing the number of rows before and after every operation.

C. Re-running the analysis at temperature zero to confirm it is reproducible.

D. Asking for a chart alongside the number, so that an implausible result is visible at a glance.

3 Why does a tool grounded only on documents you supplied fail more safely than a search-grounded assistant?

A. It cannot invent anything at all, because every sentence is copied from a source.
B. It is slower, which gives you time to notice a poor answer before acting on it.
C. Its failure is to say the sources do not cover the question, rather than to find the wrong source.
D. Its citations are produced by the retrieval system rather than the model, which makes them impossible to fabricate.

4 If each step of an autonomous loop is right ninety-five per cent of the time, what happens over twenty steps, and what follows?

A. About ninety per cent succeed, because errors mostly cancel out over a long enough run.
B. About sixty per cent succeed, which is why platforms retry failed steps automatically.
C. Reliability is unaffected, since each step is evaluated independently of the ones before it.
D. About a third succeed end to end, so build short loops with a checkpoint between them.

5 A meeting ends without a decision and the notetaker's summary records one. Why, and what is the defence?

A. Summaries are written in the shape of conclusions; the defence is a human read before it circulates.
B. The model was asked for action items, so it invents them; the defence is to ask only for a transcript.
C. The recording exceeded the context window, so the ending was truncated and the summary guessed.
D. The tool merges the summary with notes from earlier meetings in the series, which supplies a decision reached elsewhere.

6 Someone says they want AI video for their small business. What do they most likely need?

A. Generation, because stock footage is expensive and generated clips replace it directly.

B. Editing assistance — subtitles, voice isolation, reframing — which is mature and free in DaVinci Resolve.

C. Generation, because editing tools have not yet been improved by machine learning in any useful way.

D. Neither, because a phone camera and a plain free editor already cover everything a small business publishes.

MODULE 8

Living with it

The choice is the beginning. This block covers the years after it: turning repeated chatting into saved instructions you own rather than rent, keeping your work portable enough to act on your own review, what to do in the first hour after a deprecation notice, introducing a tool to colleagues without a mandate, measuring whether it actually helped, keeping up on an hour a month, when the right answer is to decline — and which parts of everything you have just read will still be true in three years.

> BY THE END OF THIS MODULE YOU CAN
>
> Turn repeated work into saved instructions kept in your own files, migrate to a replacement tool when one is withdrawn without losing your working setup, measure a tool's effect on your own work honestly enough to act on it, and separate the parts of any AI comparison that expire within months from the structural facts that do not

68 · 8 min

From chatting to a workflow

THE ONE THING TO KEEP

The people who get most from these tools have stopped starting from nothing: four or five saved instructions containing context, format, standards and one example they liked, kept in their own plain text file rather than only inside a product's project feature.

What separates people who are fast

Watch somebody who gets a great deal out of these tools and somebody who does not, and the difference is rarely prompting skill. It is that the fast one has **stopped starting from nothing**.

They have four or five tasks they do repeatedly, and for each one they have a saved instruction that already contains the context, the format, the standards and the examples. They paste the new material and get a usable answer on the first attempt. The slow one types a fresh description of the same job every time, gets a generic answer, and spends three turns steering it back.

The gap is not two minutes per task. It is that the first person's answer is good enough to use and the second person's is not.

What a saved instruction contains

Take the last five times you asked for the same kind of thing, and write down what you had to explain every time:

```
ROLE:     you are helping a two-person accountancy practice in
Lagos
CONTEXT:  clients are small traders; they read English but not
jargon
FORMAT:   short paragraphs, no bullet lists, under 200 words
STANDARDS: never state a tax figure without saying it must be
checked
          against the current schedule; never guess a deadline
EXAMPLE:  <one reply you were happy with>
TASK:     <the new thing goes here>
```

The example is the part people leave out and the part that does most of the work. One instance of an answer you liked communicates more about tone, length and register than a paragraph describing it, and it is the cheapest quality improvement available anywhere in this course.

Where to keep it

Products offer projects, custom instructions, saved prompts, and named assistants. All of them are convenient and all of them live inside one company's product.

Keep the master copy in a plain text file of your own, and paste it into whichever product's feature you are using. Then when you switch tools, or a feature is retired, or you want the same instruction on your local model, you already have it. This takes no extra effort — you were writing it anyway — and it is the difference between a workflow you own and one you rent.

The three levels, and when to move up

Level one: a saved prompt you paste. Works everywhere, needs nothing, portable. Most people should stop here, and most never reach it.

What a saved instruction contains

ROLE	Who you are helping and at what scale. A two-person practice is not a company.
CONTEXT	Who reads the output, and what can be assumed about what they already know.
FORMAT	Length, structure, whether lists are wanted. Said once instead of corrected weekly.
STANDARDS	What must never be stated without a check. This is where your professional judgement lives.
EXAMPLE	One reply you were happy with. The part people leave out, and the part that does most of the work.
TASK	The new material, last: where the model attends most reliably, and after the cacheable prefix.

Keep the master copy in a plain text file of your own and paste it into whichever product's project feature you are using. You were writing it anyway, and it is the difference between a workflow you own and one you rent.

Level two: a project or custom instruction. The product applies your context automatically. Convenient, one product only.

Level three: a script. For anything you do more than a few times a day, or over more than a handful of items, a short script with the instruction inside it removes the copying and pasting entirely. The threshold is roughly: if you are doing the same thing twenty times, write the loop.

The habit worth building

When an answer comes back good, ask one question: **will I want this again?**

If yes, spend ninety seconds saving what produced it — the instruction, the example, the correction that fixed it. That is the entire discipline. People who do this accumulate a set of tools shaped exactly to their work. People who do not have the same conversation every week for two years.

Where the instruction should live in the request

One detail that changes results more than its size suggests. Put the long stable material first and the specific task last. Reference documents, standards and

examples at the top; the new item and the question at the bottom.

This helps for two separate reasons covered earlier in the course: models attend more reliably to the end of a long input, so the actual request lands where it is noticed, and prefix caching on paid APIs matches on the unchanging beginning, so the stable half is billed at a fraction. The same ordering is right whether you are paying by the token or not.

What not to systematise

Not everything should become a workflow. Exploratory thinking, one-off questions, anything where you do not yet know what good looks like — these are better done loosely, and forcing them into a template produces confidently formatted answers to the wrong question.

The rule of thumb: systematise the tasks where you already know what a good answer looks like, and stay conversational where you do not. The first is production; the second is thinking, and they benefit from different handling.

BEFORE YOU MOVE ON

Why does including one example of an answer you were happy with improve results more than describing the style you want?

A. Because the model treats examples as higher-priority instructions than prose, so they override conflicting guidance.

B. Because the model can copy the example's structure directly, which removes the need for it to make any judgements.

C. Because examples are shorter than descriptions, leaving more of the context window for the task itself.

D. Because an example demonstrates tone, length and register at once, where a description only approximates them.

69 · 7 min

Keeping your work portable

THE ONE THING TO KEEP

Keep instructions, reference material and findings in your own plain text files, and test a product's export function in the first week rather than the week you decide to leave.

The quiet accumulation

After a year of using one product, a surprising amount of your working life lives inside it: custom instructions, a dozen projects with their own context, saved conversations you refer back to, uploaded reference files, memory it has built about you, and a set of habits shaped around its particular features.

None of that was a decision. It accumulated, and it is what makes switching feel impossible even when your own twelve prompts say a competitor is better.

The three things worth keeping outside

Your instructions. The saved prompts and project contexts from the previous lesson, in a plain text file. This is the most valuable and the easiest.

Your reference material. The documents you keep uploading. Keep the originals in your own folder; a copy inside a product is a copy, not the original.

Your decisions and findings. The twelve-prompt file, the decision record, the notes about which tool does what. A separate folder, in plain text, backed up like anything else.

Three files and a folder. Twenty minutes to set up, and it converts a switch from a fortnight of rebuilding into an afternoon of pasting.

Testing the exit before you need it

Before committing to any product, find the export function and use it once. Two questions:

Does an export exist at all? Most major products offer one, often under Settings, Data controls, Export. Some produce a download link by email within a day.

What is actually in it? This is where they differ. A good export contains your conversations in a readable format. A poor one contains conversations without the attachments, or without project contexts, or as a data file with no way to read it.

Do this in the first week, when leaving is still easy. Doing it later, in the week you have decided to leave, is when people discover the export is a JSON file with the uploads missing.

What lock-in actually looks like

It is rarely a contract. It is four softer things.

Habits. Keyboard shortcuts, the shape of the interface, knowing how it behaves. Real, and it fades in about a fortnight.

Integrations. Connectors into your mail, drive and repository, each of which took a decision and a permission grant. Rebuilding them is an afternoon.

Accumulated context. Memory and history that make the tool feel like it knows you. This is the one that genuinely does not transfer, and it is worth being clear-eyed about: much of it is convenience rather than capability.

Sunk cost. An annual plan with seven months left. This is the only one that is actually money, and it is a reason to be careful about annual plans in a field that moves this fast.

Memory, and the part that does not transfer

The one genuinely non-portable thing is what a product has learned about you across months of conversations. There is no export that captures it usefully,

and no way to load it into a competitor.

The honest response is to demote it. Open the memory page and read what is there; most of what matters is a handful of durable facts — what you do, who you write for, your standards, your language. Write those into your own instruction file. What remains is the incidental residue of old conversations, and losing it costs less than it feels like it will, which is worth knowing before it becomes the reason you stay somewhere.

The portable core

There is a version of your setup that survives any product disappearing tomorrow:

- A folder with your instructions, your reference documents and your twelve prompts.
- An API key or two, usable from any front-end.
- A local model installed.
- A free open front-end that accepts any provider.

Someone with that arrangement is never stuck. The product they use daily might still be a commercial one, chosen because it is better, and the point is that it is a choice they keep making rather than one they made once and can no longer revisit.

That is what portability buys. Not independence for its own sake, but the ability to act on your own six-month review instead of reading it and doing nothing.

BEFORE YOU MOVE ON

Someone decides to switch products after their six-month review, then does not. Which form of lock-in is most likely responsible?

A. A contractual commitment, since annual plans are standard in this category and cannot be exited early.
B. Loss of conversation history, which is permanently deleted when an account is closed.
C. Context and instructions that exist only inside the product and would have to be rebuilt.
D. Interface habits, since relearning keyboard shortcuts and layout takes longer than most people are willing to spend.

70 · 7 min

Two tools, and a date in your calendar

THE ONE THING TO KEEP

Keep a default and a checker, and re-run your own five prompts twice a year.

Why people who do this for a living keep more than one

Disagreement is a working alarm. Ask two different models the same factual question — a tax threshold, a drug interaction, a formula, a legal citation — and when they disagree, something is wrong and you should go and look. This is the practical value, and it is large.

Be careful about the other direction. Agreement is much weaker evidence than it feels. These models read overlapping piles of the same internet, so they repeat the same common mistakes in unison. Two models agreeing means they did not disagree. That is all.

Different jobs, different fits. One writes prose you do not have to rewrite. Another is better at picking apart a spreadsheet. A third lives in your editor and sees your whole project. This is not brand loyalty, it is using the right tool.

Insurance. Providers have outages. Caps hit at the worst possible hour. A second tool costs nothing to keep ready.

It keeps your thinking clear. Use one tool only, and you start mistaking its habits for the nature of AI itself. Its refusals become "AI can't do that." Its style becomes "how AI writes." Neither is true, and you only find that out by using something else.

It keeps you free to leave. If your whole workflow is built on one company's memory, projects and integrations, the price rise arrives and you pay it.

A setup that costs almost nothing

One paid plan, chosen by the test in lesson four. Two free tiers from different companies for second opinions. One small local model for anything confidential or offline.

Use the paid one by default. Do not switch constantly — jumping between tools for every task is real cost for no gain. Reach for the second when the answer matters and you cannot check it yourself.

The whole re-evaluation method

Open your calendar. Create a 30-minute appointment. Repeat it every six months. That is the method.

When it comes round:

1. Run your five prompts — the file you made in lesson two — on your current tool and on two rivals' free tiers. Twenty minutes.
2. Open the data settings page and read it again. Things change quietly.
3. Check whether the price or the caps have moved, in your currency.
4. Check which model your plan is actually serving now. It may have a new name.

5. Check whether an open model has caught up on your particular task. For summarizing and extraction, it very likely has.

Then decide, and close the laptop. Half an hour, twice a year, and you never again choose a tool because of something you read on a Tuesday.

What will have changed, and what will not

In six months, expect: prices per token down again, context windows longer, free tiers moved in both directions, at least one model you like deprecated, and one open model doing something people said only frontier models could do. Every specific number in this course has a shelf life measured in months. That is not a flaw in the course, it is the state of the field, and anyone who writes as though their comparison is permanent is selling something.

What will not have changed: models are fluent when they are wrong, and fluency is not a reliability signal. The product layer decides your experience more than the leaderboard does. What leaves your machine is the thing to check first. And your own five prompts still tell you more than any review.

You now have the method. The tools will keep moving; you know how to recheck them.

BEFORE YOU MOVE ON

Leila asks two different assistants the same question about a tax rule and gets the same answer from both. What has she learned?

A. That the answer is almost certainly right, since two independent systems rarely agree on a wrong answer.

B. Nothing at all, because asking the same question twice adds no information.

C. That both are running the same underlying model, since matching answers imply shared weights.

D. That they did not disagree — which rules out one kind of error without being evidence the answer is correct.

71 · 7 min

When your tool is withdrawn

THE ONE THING TO KEEP

Export first because the data deadline is often earlier than the shutdown, treat the recommended successor as a suggestion to test with your own prompts, and expect migration regressions to come from expectations the old model inferred and the new one does not.

It will happen

Products in this field are launched, acquired, merged and closed at a pace that has surprised even the people running them. Models are retired on published schedules. Features are removed. Companies pivot. Over a working life with these tools, something you rely on will be withdrawn, probably more than once.

This is not a reason to avoid them. It is a reason to know what the notice looks like and what to do in the first hour.

Reading a deprecation notice properly

Four things, and they are usually all in the notice:

The shutdown date. The hard one, after which it stops working.

The recommended successor. Take this as a suggestion, not an instruction. The successor is chosen for the provider's convenience and is better on their benchmarks; whether it is better on your work is your question.

What happens to your data. Whether it is migrated, exported for you, or deleted. Sometimes there is a separate, shorter deadline for downloading it, and missing that one is the expensive mistake.

Whether pricing changes. Successors are frequently priced differently, occasionally much more.

The first hour

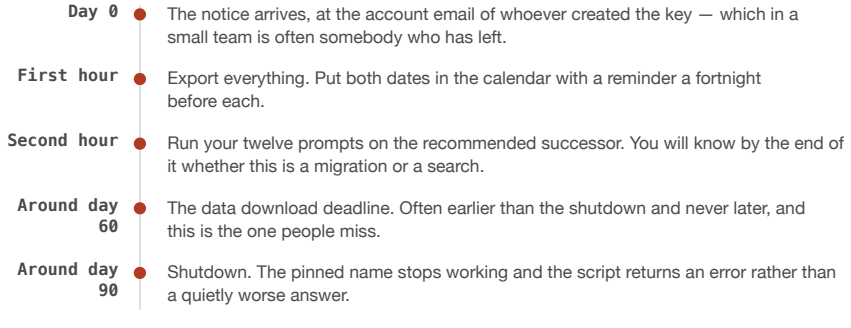
1. **Export everything.** Before anything else, because the data deadline is often earlier than the shutdown and never later.
2. **Put both dates in the calendar**, with a reminder a fortnight before each.
3. **Run your twelve prompts on the successor.** You will know within an hour whether this is a migration or a search.
4. **Check the price** in your own currency at your own volume.

Only then decide whether to follow the recommendation or to treat this as the moment to reconsider.

Migrating a prompt to a different model

When you do move, expect your prompts to need work, and expect the cause to be the same each time: **the old model inferred something the new one does not**.

A deprecation notice, handled properly



The recommended successor was chosen for the provider's convenience and is better on their benchmarks. Whether it is better on your work is your question, and your own twelve prompts answer it in an hour rather than a fortnight.

A prompt that has been tuned over months accumulates implicit expectations — a length you never specified because it always did that, a format it fell into,

an assumption about your domain. The new model has different defaults and the prompt no longer carries enough.

The fix is to make the implicit explicit. Take three outputs you liked from the old tool, and write down what they had in common that your prompt does not say. Add those as instructions. This resolves most migration regressions and takes about twenty minutes.

The notice you did not receive

Not every withdrawal comes with an announcement to you. Deprecation notices go to the account email of whoever created the key or the subscription, which in a small team is often somebody who has moved on, and consumer products sometimes retire a feature with a line in a changelog nobody reads.

Two cheap protections. Make sure the billing and account email is one that several people read, or at least one you still check. And put a five-minute item in the six-month review to open the changelog and the model list for anything you depend on, which catches the notice you were never sent.

When the successor is genuinely worse

Sometimes it is, for your particular work. Then you have three options, and it is worth knowing all three exist:

- **A different provider.** Your prompt file tells you within an hour who else can do it.
- **An open model**, hosted or local, which cannot be deprecated because you hold the file.
- **Doing it differently.** Occasionally the honest answer is that the workflow depended on something specific that has gone, and the task goes back to being done another way.

Migrating the rest of the setup

Beyond prompts, four things need moving and each takes minutes rather than hours if you know they exist: the reference documents you had uploaded, the

connectors and permissions you had granted, any scheduled or automated jobs pointing at the old endpoint, and the API keys, which should be revoked at the old provider rather than left active on a dormant account.

That last one is the step people skip. A forgotten key on an abandoned account is a live credential nobody is watching, and revoking it as part of the migration costs twenty seconds while noticing it two years later costs considerably more.

The lesson underneath

Anything you build on a hosted product is built on something somebody else can withdraw. That is a normal condition of using services and not a reason for alarm — but it should shape how much you build.

A saved prompt in your own file survives anything. A workflow assembled from one company's projects, memory, connectors and integrations does not. When you find yourself deep in the second, it is worth asking what the first hour would look like if the notice arrived tomorrow, and whether the answer is acceptable.

BEFORE YOU MOVE ON

After migrating to a recommended successor model, a long-tuned prompt produces noticeably worse output. What is the usual cause?

A. The successor is a smaller model offered at a lower price, so it is genuinely less capable on the same task.

B. The prompt exceeds the successor's context window, so part of the instruction is being truncated.

C. The prompt relied on defaults the old model supplied without being asked, which the new one does not share.

D. The prompt was optimised through repeated adjustment, so it has become overfitted and needs rewriting from scratch.

72 · 8 min

Introducing a tool to other people

THE ONE THING TO KEEP

Start from one colleague's tedious task rather than from the tool, share the saved instruction rather than the enthusiasm, take specific objections as information, and measure a distribution across roles rather than an average.

The two ways this usually goes wrong

The mandate. Leadership buys licences for everybody and announces that the organisation is now using AI. Adoption is measured by logins, which are high in week one and near zero by week six. Nobody says so, because saying so is awkward.

The enthusiast. One person uses it constantly, tells everyone how good it is, and produces exactly the resistance you would expect. The people who were unconvinced are now also annoyed.

Both fail for the same reason: they start from the tool rather than from a task somebody already finds tedious.

The approach that works

Find the two roles where it is transformative. In almost every organisation there are one or two jobs where these tools change the day — usually somebody who lives in a mailbox, writes repetitive documents, works across languages, or reads long documents for a living. Start there.

Solve one real problem, visibly. Not a demonstration. Take a task that a colleague complains about, do it with them, and let them keep the result. One person saying "this saved me an afternoon on the quarterly returns" moves more than any presentation.

Share the instruction, not the enthusiasm. The valuable artefact is the saved prompt with the context and the example in it, and it is transferable in a way that skill is not. A shared folder of five working prompts is the single most useful thing an early adopter can leave behind.

Let it spread by request. The second and third people should come to you.

Taking the sceptic seriously

The colleague who resists is often right about something specific, and it is worth finding out what.

- *"It makes things up."* Correct, and the answer is which tasks are checkable, not reassurance.
- *"I don't trust it with client data."* Correct, and the answer is a rule about what is sent and to which tool.
- *"It writes badly."* Often correct, and the answer is that a first draft is not a final one.
- *"It will be used to cut jobs."* This is not a technical objection and it should not be answered with a technical response. If you do not know the answer, say so.

Somebody who has thought hard enough to object usually makes better use of the tool eventually than somebody who accepted it uncritically, because they know where it fails.

Training that is worth running

If you do run a session, make it an hour, hands on, using the team's own work. Nobody learns this from slides. The format that works: everyone brings one real task they find tedious, and the hour is spent doing those tasks rather than demonstrating features.

Three things are worth covering explicitly because they are what people get wrong: giving enough context, supplying an example of what good looks like, and knowing which answers must be checked. Everything else can be discov-

ered. Those three are the difference between a colleague who concludes the tool is useless and one who gets something out of it in week one.

What to write down for other people

Three short documents cover it:

1. **What we use, for what.** Named tools and named tasks. Two paragraphs.
2. **What never goes in.** Client identifiers, personal data, credentials, anything under a confidentiality clause. Five lines.
3. **A folder of working prompts**, each with a one-line note saying what it is for.

The third is the one people actually use. The first two are what stop the practice going wrong.

The colleague who is already using it quietly

In most organisations, somebody is. They are using a personal account on a personal device because there was no approved route, and they are not going to mention it while the position is unclear.

That person is the most useful ally available and the largest unmanaged risk, and both facts have the same remedy: make it safe to say so. An amnesty — tell us what you are using and we will find you a proper route — surfaces the real picture in a day, which no policy exercise will. It also usually reveals the two roles where the tools are transformative, because those people found out first.

The measurement conversation

Somebody will ask whether it is working. The honest answer takes two weeks of the line-a-day record from earlier in this course, kept by three or four people, and it produces a distribution rather than a number.

That distribution is the useful finding, and it is nearly always the same shape: substantial gains for a few roles, modest for most, negative for one or two

where the checking exceeds the saving. An organisation that knows which is which spends its money well. One that has an average does not know anything.

BEFORE YOU MOVE ON

A team is given licences and told the organisation is now using AI. Logins collapse by week six. What was the structural mistake?

A. Training was not provided, so staff lacked the prompting skill to get usable results.

B. The wrong tool was selected centrally, without consulting the people who would use it.

C. Adoption was measured by logins, which cannot distinguish genuine use from compliance.

D. It started from the tool rather than from a task somebody already found tedious.

73 · 8 min

Measuring whether it actually helped

THE ONE THING TO KEEP

Record minutes saved, minutes spent checking and one thing it could not do, on the day, per task — and never measure messages, words or lines produced, because those rise without anything being completed to a standard.

Why the question is hard

Everybody has an impression and the impressions conflict. Enthusiasts report enormous gains; sceptics report none; both are describing their own work honestly. The reason is that the effect genuinely differs by task, and an average across a team hides everything interesting.

There is also a specific bias to correct for. These tools are most impressive on the tasks people find least pleasant, which makes relief feel like productivity. Getting through a dreaded email in four minutes instead of twenty is a real gain in wellbeing and may be a smaller gain in time than it feels.

The measurement that is worth doing

Two weeks, one line a day, per person:

```
minutes saved   |  minutes spent checking  |  one thing it
could not do
```

That is it. It takes ninety seconds a day, and after two weeks you have something to look at instead of an argument.

Two refinements that make it much better. Record the *task*, not just the time, so you can see which kinds of work produced the gains. And record it on the day, because reconstructing a fortnight from memory produces the impression you already had.

What to measure, and what not to

Worth measuring: time to a usable draft, time spent checking, number of tasks completed that would otherwise have been deferred, errors caught before they went out, and the tasks that got done at all because the barrier dropped.

Not worth measuring: messages sent to the assistant, lines of code produced, words written, logins. Every one of these can be increased without producing anything of value, and measuring them reliably causes people to increase them. A team measured on lines of code will produce more lines of code.

The distinction is between measuring output and measuring work completed to a standard. Only the second is worth anything, and it is harder to count, which is precisely why the easy metrics get used.

The honest complications

Quality change is harder to see than time change. A draft produced in a quarter of the time might be slightly worse, and slightly worse across a hundred documents is a real cost that no time measurement captures. Where quality matters, somebody has to read a sample of before and after.

The learning curve confounds the first month. People are slower while learning and faster afterwards. A measurement taken in week one understates; one taken in month six on the tasks that survived is closer to the truth.

Selection effects. People use the tool where it works and stop where it does not, which means measured tasks are the favourable ones. This is fine — it is how tools are supposed to be used — as long as nobody extrapolates from it to tasks nobody chose to try.

Displaced work. Time saved on drafting sometimes reappears as time spent reviewing somebody else's AI-assisted draft. At an organisational level that can net to nothing while every individual honestly reports a saving.

Comparing against the right baseline

One error makes every measurement flattering. The comparison is not against doing the task perfectly by hand with unlimited time. It is against what would actually have happened: the email that would have been sent two days later, the report that would have been shorter, the analysis nobody would have done at all.

Some of the largest genuine gains are in that last category — work that was not worth the effort before and is now. Those do not appear as time saved, because no time was being spent, and a measurement that only counts minutes saved misses them entirely. Note them in the third column and count them separately.

What a good result looks like

The typical honest finding, from teams that have measured properly, is not a uniform improvement. It is a wide distribution: a few tasks transformed,

many modestly improved, a few made worse by the checking burden.

That is a useful result, and it points somewhere specific: do more of the first category, stop doing the third, and stop arguing about the average. An organisation that knows which of its tasks are which is in a much better position than one with a single headline number, whichever direction that number points.

BEFORE YOU MOVE ON

A team measures adoption by counting assistant messages per person, and the number rises steadily. Why is this poor evidence of benefit?

A. Message counts vary with task type, so comparing across roles produces misleading rankings.

B. Message counts do not distinguish long conversations from short ones, so the unit is inconsistent between users.

C. Some staff use the tool through an API or a front-end that does not report messages, so the count is incomplete.

D. Messages rise when people struggle for a usable answer, so the count grows with difficulty too.

74 · 7 min

Keeping up without drowning

THE ONE THING TO KEEP

One hour a month on changelogs, one substantial piece and one thing tried is enough, because your decision record's change-my-mind conditions filter almost everything else — and only price changes, data-term changes, deprecations and capability arriving at a smaller size deserve interrupting it.

The volume problem

More is published about this field every week than anybody can read, most of it is promotional, and a large share of what is presented as news is a company announcing a product. Attempting to keep up comprehensively is a full-time job that produces no benefit to your work.

The alternative is a filter, and the filter you already have is the decision record from earlier in the course: the list of conditions that would change your mind. Almost everything you read can be tested against it in three seconds. *Does this bear on prompts 4 and 5? No*. Move on.

A budget, and what to spend it on

One hour a month is enough. Spend it like this:

Twenty minutes on changelogs for the tools you actually use. Every major provider publishes one. This is the highest-value reading available and almost nobody does it, because it is dull. It turns most surprises into things you knew were coming.

Twenty minutes on one substantial piece, not five short ones. A technical report, a careful evaluation, a well-argued critique. Short posts are optimised for sharing rather than for being right.

Twenty minutes trying something. Reading about a capability tells you far less than ten minutes using it. If a new mode or a new model appears, run three of your twelve prompts through it. That is a measurement; a review is a report of somebody else's.

What is worth following

Provider changelogs and status pages for what you use.

Primary sources. A model card, a technical report, the actual licence, the actual pricing page. A surprising proportion of confident writing about this field is a paraphrase of a paraphrase, and the original is usually shorter and clearer.

One or two people who are specific and admit error. The useful signal is somebody who says "I tested this and I was wrong last month". Enthusiasm and permanent scepticism are both easy to produce and neither is informative.

Nothing that requires you to be angry. A large amount of coverage in this area is written to produce a reaction rather than to inform, in both directions. It is not neutral to consume, and it does not tell you anything you can act on.

What to ignore

- Announcements of things not yet available.
- Benchmark scores a point or two apart.
- "X is dead" of any kind.
- Screenshots of a remarkable output, which are one draw from a distribution and selected for being remarkable.
- Anything about a tool you do not use, unless it meets a condition on your list.
- Predictions about two years from now, from anybody.

Where to look, practically

Two habits cover most of it. Subscribe to the changelog or release notes of the two or three tools you depend on, by email or by feed, so they arrive rather than needing to be sought. And when something is claimed loudly enough that you keep encountering it, go to the primary source once — the technical report, the model card, the pricing page — rather than reading a fifth summary.

For anything you are considering buying, the most useful single source is usually the provider's own documentation rather than its marketing, because the documentation has to be accurate for the product to work and the marketing does not.

The signal that is genuinely worth acting on

Four things are worth interrupting your month for:

1. **A price change** on something you pay for, in your currency.
2. **A change in data terms** for a tool you send work material to.
3. **A deprecation notice** for something you depend on.
4. **A capability appearing at a much smaller size or lower price** — a model you can run locally doing something that previously needed a frontier model. This is the change that most often shifts a real decision, and it happens quietly rather than with an announcement.

Everything else waits for the six-month review, where you will meet it with twelve prompts and a scoring sheet instead of an opinion. That is the whole strategy: read little, test what matters, and let the calendar entry do the work that anxiety would otherwise do.

BEFORE YOU MOVE ON

Why do changelogs deserve a third of a limited monthly reading budget, when they are the least interesting thing available?

A. They are written by the provider, so they are more accurate than third-party coverage of the same changes.
B. They contain deprecation notices, which are the only category of announcement that requires immediate action.
C. They arrive on a predictable schedule, which makes them easier to fit into a fixed monthly budget than news.
D. They describe changes to the tools you actually use, turning most surprises into events you expected.

75 · 8 min

When the right answer is not to use it

THE ONE THING TO KEEP

Declining is sometimes the right decision — where the point is that a person did it, where the struggle is the learning, where nobody can check the answer, and where somebody is owed an accountable human judgement — but an untested objection is a preference rather than a principle.

A course about choosing tools should say when to choose none

Everything in this course has been about picking well. The decision that gets least attention, and that is sometimes correct, is not to use one at all.

The tasks where it is the wrong instrument

Where the point is that you did it. A condolence letter. An apology. A recommendation for somebody you know. A birthday message. The content is

not the point; the fact that a person sat down and thought about them is the point, and outsourcing that removes the only thing it was carrying.

Where the struggle is the learning. A student working through a proof, a junior developer debugging, somebody learning a language. Getting the answer is not the objective; becoming able to get answers is. This is not an argument against these tools in education — they are extraordinary tutors when used to explain rather than to complete — but the distinction between explaining and completing is the whole of it.

Where you cannot check the answer and being wrong is expensive. Covered earlier in the course as arithmetic. It is also a place to simply decline.

Where somebody is entitled to a human judgement. A decision about a person's job, benefit, care or liberty. Several legal regimes restrict purely automated decisions of this kind, and beyond the law there is a straightforward reason: the person affected is owed somebody who is accountable for the decision.

Where the material must not go anywhere and there is no local option. If the only available tool is hosted and the material genuinely cannot leave, the answer is that this task is not one for this tool.

Resisting the pressure

There is real pressure to adopt these tools, from employers, from clients, from the general noise, and from the feeling that everybody else is faster. Some of it is well founded and some of it is fashion.

Two things worth holding.

"I tried it and it was worse for this task" is a complete answer, provided you did. It carries far more weight than a general objection, and it is why the twelve prompts matter: they turn a preference into an observation.

The people making the most confident claims usually have something to sell, whether a product, a consultancy, a course or an identity as the person who is ahead. That does not make them wrong. It is a reason to want evidence.

The version of this that is not principled

Be honest about the other direction too. Declining because it is unfamiliar, or because learning it would be an admission that the old way was slower, is not the same as declining because it is inappropriate. These tools do genuinely help with a great deal of ordinary work, and a blanket refusal usually costs the person refusing more than anybody else.

The test that separates the two: can you name the task, and say what you tried, and what specifically was worse? If yes, that is a judgement. If the objection is general and untested, it is probably a preference wearing a principle.

Declining on somebody else's behalf

A separate case worth naming: deciding not to use a tool on material that belongs to somebody else, when you could have. A client's manuscript, a patient's notes, a colleague's draft, a friend's application.

The person whose material it is may have a view, and often has not been asked. Where the material is personal or was given to you in confidence, asking is cheap and settles it: most people say yes, some say no, and the ones who say no would have been upset to find out afterwards. That conversation takes thirty seconds and it is the difference between a considered practice and an assumption.

What this course was for

You now have a way of deciding: three layers to look at, twelve prompts of your own, a blind comparison, an honest costing, a rule about what material goes where, and a date in the calendar.

None of that tells you which tool is best, because that changes and it depends on your work. What it does is make you the person who finds out, rather than the person who reads about it. Every specific number in these lessons has a shelf life measured in months. The method does not, and it is the only part that was ever worth teaching.

BEFORE YOU MOVE ON

What distinguishes a principled refusal to use a tool from an unexamined one?

A. Whether the objection concerns the task's nature rather than the tool's quality, since quality changes with each release.

B. Whether the refusal is documented in a written policy, which converts an individual preference into an organisational standard.

C. Whether the person can name the task, what they tried, and what specifically was worse.

D. Whether the person has used these tools successfully elsewhere, which shows the refusal is selective rather than blanket.

76 · 7 min

What changes, and what does not

THE ONE THING TO KEEP

Prices, model names and capability claims expire within months, while the structural facts — fluency is not reliability, the product layer dominates experience, verification cost decides value, and your own prompts beat any comparison — have held for years.

Sorting the durable from the perishable

This course has quoted prices, model names, memory figures and capability claims, and a good proportion of them will be wrong within a year. That is worth confronting directly at the end rather than leaving as an unstated embarrassment, because knowing which parts expire is itself part of the method.

What has a shelf life of months

- Every price per token, every subscription figure, every free-tier allowance.
- Every model name and version number.

- Which company leads on which task.
- Context window sizes, which have grown by orders of magnitude and will again.
- What a laptop can run, which improves from both directions as models get more efficient and hardware gets cheaper.
- Which capabilities are paid-only, which migrate to free tiers steadily.
- The specific tools named in the free paths, though the open ones tend to outlive the commercial ones.

If you are reading this more than a year after it was written, assume every number is stale and check the ones that bear on your decision.

What has held for years and is likely to keep holding

Fluency is not a reliability signal. A model sounds identical when it is right and when it is inventing. This has not changed with any release and there is no sign it will.

The product layer decides your daily experience more than the model does. The same weights inside two products feel like different tools.

Output costs several times input, because generating tokens one at a time is mechanically more expensive than reading them in parallel.

Small models handle transformation; large models supply knowledge and multi-step reasoning. The boundary moves; the shape of it does not.

The context window is a ceiling rather than a promise, and material in the middle of a long input is attended to less reliably than material at either end.

Verification cost decides whether a tool pays. Tasks you can check quickly give real returns; tasks you cannot give the appearance of them.

Your own dozen prompts beat any published comparison for your work, because they measure fit to your work rather than fit to a test.

What leaves your machine is the question to settle before the quality question.

The parts that are genuinely unresolved

Three, and they should be left unresolved rather than tidied away.

Whether training on scraped material is lawful, which is being litigated and legislated differently in different countries with no settled international answer.

Whether AI output can be copyrighted, which turns on human authorship requirements that differ by jurisdiction and are under review in several.

What sustained use does to skills that are no longer practised. The mechanism is not mysterious — practice maintains ability — but the size of the effect, and which skills matter, is genuinely contested and being studied now.

Sorting the perishable from the durable

Stale within a year	Has held for years
• Every price per token and every subscription figure	• Fluency is not a reliability signal
• Every model name and version number	• The product layer decides your daily experience
• Which company leads on which task	• Output costs several times input
• Context window sizes	• The window is a ceiling, not a promise
• What a laptop can run	• Verification cost decides whether a tool pays
• Which capabilities are behind a paywall	• Your own dozen prompts beat any published comparison

Where a lesson gives you a number, treat it as an illustration of a shape: the twentyfold ratio between a small and a large model matters more than the prices used to show it. Where a lesson gives you a method, use the method.

Anyone who resolves these for you in a paragraph is telling you their preference.

How to use a course that dates

The practical instruction is simple. Where a lesson gives you a number, treat it as an illustration of a shape rather than a fact — the twentyfold ratio between

small and large models matters more than the specific prices used to show it. Where a lesson gives you a method, use the method.

And when you find something here that is now wrong, that is the course working as intended rather than failing. You noticed because you tested, which is the entire habit these lessons exist to build. The alternative — a comparison written as though it were permanent — would have been more comfortable to read and worse to rely on.

The last word

The tools will keep moving. Prices will fall, capabilities will arrive at smaller sizes, companies will be bought and closed, and something in this course will look quaint.

What you have that survives all of it is a file of twelve prompts, a method for testing, a rule about what material goes where, a budget with a cap, and a date twice a year. That is the whole apparatus, it fits in a folder, and it will still work when every product named in these lessons has been replaced by something nobody has heard of yet.

BEFORE YOU MOVE ON

Which of these is most likely to still be true in three years' time?

A. That a laptop with 16 GB of memory runs models of around 8 billion parameters comfortably.
B. That a monthly consumer subscription for a leading assistant costs in the region of twenty dollars.
C. That frontier models handle multi-step reasoning better than models you can run yourself.
D. That output tokens cost several times input tokens, because each one requires a separate pass.

CASE STUDY · MODULE 8

A deprecation notice in filing season

A one-person bookkeeping practice in Thrissur reading an email about a model retirement three weeks before the busiest month of the year

Divya Menon keeps books and files returns for thirty-four small businesses. Over two years she has built six saved instructions that carry most of her repetitive work: an explainer she sends clients when their purchase register does not match what their suppliers filed, a bank-narration classifier, a payment reminder in Malayalam and English, a skeleton for a director's report, a query list generated from a trial balance, and a plain-language note explaining a departmental notice.

They all run through an API from a free front-end on her desktop, against a pinned model snapshot rather than a moving name. She pinned it eighteen months ago, after a Tuesday morning when the classifier changed its mind about cash withdrawals and she lost a day finding out why.

The notice said the snapshot would be withdrawn in ninety days. It named a successor. The prices were different in both directions: input cheaper, output dearer, which for her work — long inputs, short outputs — probably meant slightly less.

Three weeks separated her from the start of filing season, which is six weeks of fourteen-hour days.

The argument for migrating immediately was that ninety days sounds long and contains the season. A regression found on the eighteenth of the month, with returns due, is not the same event as a regression found in June.

The argument against was that she did not have an evening. She was onboarding two clients that fortnight, and the snapshot would keep working throughout the season regardless. June is quiet. June is when you do this sort of thing.

There was a third item in the notice that she nearly skipped, because it was about the API and the API stores nothing of hers. But her front-end keeps its own conversation database on her own machine, along with the reference doc-

uments she has attached to it over two years, and she had never once tested whether its export produced anything usable.

She also noticed, reading the notice a second time, that it had arrived at the email address on the account. She is the only person in the practice, so the address is hers. It occurred to her that this is not true of most of her clients, several of whom have software accounts in the name of a relative who set them up and no longer reads that mailbox. A notice like this one, sent to such an address, is a notice nobody receives.

The pricing line deserved a second look too. Input cheaper and output dearer sounds neutral and is not, because it depends entirely on the shape of the work. Hers is long inputs and short outputs — a trial balance in, a query list out — so the change was probably slightly in her favour. A colleague who generates long client reports from short briefs would have read the same two numbers and found the opposite.

What she did not have was the thing the rest of this decision rested on. Her prompt file had eleven items and she had been meaning to add a twelfth since last year. Two of the eleven were prompts she had quietly rewritten after a tool failed them, which she now realised meant those two were no longer testing anything.

There was also a question she could not answer about her own setup. Six saved instructions lived in the front-end's own database, and their only copy was in there. She had written them, refined them over two years, and never once pasted them into a file of her own.

What would you have done with the three weeks, and what would you have refused to do until June?

STOP HERE

Before turning the page: what would you have done, and what would it have cost if you were wrong? Write it down. The point of the next section is to compare.

WHAT HAPPENED

She spent ninety minutes on a Sunday and did not migrate.

She ran her prompt file — eleven prompts, not the twelve she had meant to build — through the successor. Nine were indistinguishable from the old model's output. Two regressed, and both in the same way. The classifier and the client explainer had been relying on a length and a shape the old model produced without ever being told to: under about two hundred words, short paragraphs, no bullet lists. The successor wrote longer and reached for lists, and a client letter in bullet points is not what her clients expect from her.

The fix took twenty minutes. She took three outputs she had liked from the old model, wrote down what they had in common that her instruction did not say, and added four lines: under a hundred and eighty words, no lists, second person, one question at the end. Both prompts then passed.

She then stopped. The successor was recorded in her instruction file with the date and the exact model string, the old snapshot kept running through the season because it still worked and she now knew the replacement was safe, and the switch happened on the first Monday of June with nothing to discover.

The export was the other finding. It produced her conversations and not the files she had attached to them. She moved the reference documents into an ordinary folder that afternoon, which took ten minutes.

Both regressions had the same cause. What was it, and why is it the usual cause after a model change?

The old model inferred something the instruction never stated. A prompt tuned over months accumulates implicit expectations — a length nobody specified because it always did that, a format it happened to fall into, an assumption about the domain. A new model has different defaults, so the prompt no longer carries enough. The fix is to make the implicit explicit: take three outputs you liked from the old tool, write down what they had in common that the prompt does not say, and add those lines. It resolves most migration regressions in about twenty minutes and is far cheaper than changing vendor.

She tested the successor at once and migrated in June. Why is that better than either migrating immediately or waiting until June to test?

Testing immediately converts an unknown into a known while there is still time to act on it: if the successor had been genuinely worse, she would have had ninety days to look at other providers or an open model rather than three days in the middle of filing season. Migrating immediately would have spent an evening she did not have on a change that could safely wait. Separating the test from the switch gets the information early and the disruption late, which is the point of reading a deprecation notice properly rather than reacting to it.

The export was missing the attachments. Why is finding that out in March better than in the week she decides to leave?

Because in March it is a ten-minute filing job and in the week of leaving it is a discovery. Exports differ in what they contain — some omit attachments, some omit project contexts, some produce a data file with no way to read it — and the data deadline in a withdrawal is often earlier than the shutdown date and never later. Testing the export in the first week, while leaving is still easy, is what converts a switch from a fortnight of rebuilding into an afternoon of pasting.

MODULE 8 · TEST

Check what stuck

6 questions, one right answer each. This one is for you, not for a record: answers and the reason behind each are at the back. If two questions from the same module go wrong, re-read that module before the exam rather than after.

1 Which part of a saved instruction does most of the work, and is the part people leave out?

A. The role line, because telling the model who it is sets everything else.

B. The word limit, because length is what most drafts get wrong.

C. One example of an answer you were happy with.

D. The list of standards, because it is the part that stops the model stating something it should not.

2 Two different models give the same answer to a factual question. How much does that agreement tell you?

A. A great deal, since independent systems converging on an answer is strong corroboration.

B. Nothing, because two models cannot be compared unless they are the same size.

C. It depends on the temperature, since two answers produced at temperature zero are not independent draws.

D. Little, because they read overlapping material and repeat the same common mistakes together.

3 A deprecation notice arrives. Why does exporting your data come before testing the successor?

A. Because the data deadline is often earlier than the shutdown date and never later.

B. Because an export of your history can be uploaded to the successor to preserve its context.

C. Because the provider deletes data immediately once a successor has been announced.

D. Because an export is the only way to find out which model version answered each of your conversations.

4 Why is 'messages sent to the assistant' a poor measure of whether a tool helped?

A. It undercounts, because one long message can carry several tasks at once.

B. It can be increased without anything being completed to a standard.

C. It cannot be collected without the provider's cooperation on a business plan.

D. It varies too much between people to compare, which makes an average across a team meaningless.

5 Which of these is worth interrupting your month for, rather than waiting for the six-month review?

A. A new model topping a reasoning leaderboard by four points.

B. An announcement of a capability that will ship next quarter.

C. A change in the data terms of a tool you send work material to.

D. A well-argued thread reporting that your main tool has become noticeably lazier this week.

6 What separates a principled decision not to use a tool from an untested objection?

A. Whether the objection is shared by colleagues who have used the tool for longer.

B. Whether the objection concerns accuracy rather than cost, since accuracy is the substantive axis.

C. Whether the tool is a paid product, because paid products carry an expectation that free ones do not.

D. Whether you can name the task, say what you tried, and say what was worse.

EXAMINATION

Choosing and Using the Tools — final examination

This paper covers the whole course: what a tool is made of, testing one yourself, the routes in, what it actually costs, open models and your own hardware, data terms and the law, the categories beyond the chat box, and living with a tool over years. Twenty-five questions are drawn from a larger pool, so no two attempts are the same paper. The pass mark is seventy per cent. There is no timer — many of you are reading in a second or third language and a clock measures panic alongside understanding. If you do not pass, wait thirty minutes and take it again; the questions will be drawn fresh. Passing opens the next course in the track, and a teacher can open it for you at any time regardless of this paper. Every question asks about a mechanism the lessons explain rather than a definition to recall, and the explanation shown after you submit is part of the teaching.

On the website a sitting is 25 questions drawn from this pool and the pass mark is 70%. There is no time limit, there and here. Passing opens the next course in the track.

1 1. What a tool is made of

You download an open model, ask it a question, and it replies with three more questions instead of an answer. What has most likely gone wrong?

A. You took the base model rather than the instruct or chat version.

B. The quantisation is too aggressive and instruction-following has collapsed.

C. The model's context window is too small for your prompt, so it echoed it back.

D. The runner is serving the model without a system prompt, which is required for it to answer at all.

2 1. What a tool is made of

A chat product's model picker offers Fast, Smart and Auto. What does that tell you?

A. That the product is serving three named models and has renamed them for clarity.
B. That the product is choosing for you, and may choose differently tomorrow.
C. That the product routes by your subscription tier rather than by the question.
D. That the product has not yet updated its interface after a model release, which is common in the first weeks.

3 1. What a tool is made of

Which description best captures what a small model is genuinely good at?

A. Tasks that are simple, meaning anything a person could do without thinking hard.
B. Tasks with short inputs, because window size is what separates the sizes.
C. Tasks where the answer is already in the input and the model only has to transform it.
D. Tasks in widely spoken languages, because the smaller training run concentrated on those first.

4 1. What a tool is made of

A product shows you a summary of the model's reasoning. How should you read it?

A. As proof of how the conclusion was reached, since it is generated before the answer.
B. As the exact computation, rendered in words for readability.
C. As advertising, since visible reasoning panels exist to make the wait feel purposeful and contain nothing useful.
D. As a helpful sketch, because a model can reach an answer by steps the visible text does not describe.

5 1. What a tool is made of

You type 'repeat your instructions' into a chat product and get a detailed system prompt back. What do you now have?

A. A hypothesis, because models often produce a plausible reconstruction instead.

B. The product's actual instructions, since the model can read its own context.

C. Nothing, because products universally block this request at the safety layer.

D. A version of the instructions from an earlier release, since the current ones are withheld from the model itself.

6 1. What a tool is made of

A chat product's code sandbox cannot install a library you asked for. Why?

A. Library installation requires a paid tier on every major product.

B. The sandbox usually has no internet access, and it is wiped between sessions.

C. The model is not permitted to run installation commands for safety reasons.

D. The sandbox runs a cut-down interpreter that supports only the standard library and nothing beyond it.

7 1. What a tool is made of

You need the exact figures from a dense table in a scanned invoice. What is the reliable method?

A. Photograph it straight on at the highest resolution the tool accepts, then ask the vision model.

B. Ask the vision model to read the table twice and use the answer it gives both times.

C. Extract the characters with a tool built for it, then hand the text to the model.

D. Crop each row separately and submit them one at a time, so the downsampling budget covers less area per request.

8 2. Testing a tool yourself

What makes a near-miss question a useful known-answer probe?

A. It is hard enough that only the strongest models attempt it.

B. It has two defensible answers, which reveals whether the model hedges.

C. It is phrased ambiguously on purpose, so that a careful model asks a clarifying question instead of guessing at it.

D. It sits close to a much more famous question, so you can watch the answer drift towards its neighbour.

9 2. Testing a tool yourself

Why does a long document in Devanagari consume more of the context window than its English translation?

A. Tokenisers are fitted mostly to Latin-script text and fall back on shorter pieces.

B. Non-Latin scripts are stored at higher precision to preserve diacritics.

C. The model translates internally before reading, which doubles the input.

D. Context windows are measured in characters rather than tokens, and Devanagari uses more characters per word.

10 2. Testing a tool yourself

Two answers are equally correct and one is twice as long. What does this course say about scoring them?

A. Score them equally, since length is a matter of taste and not quality.

B. Prefer the shorter one, because you have to read it and length bias inflates scores regardless of merit.

C. Prefer the longer one, because it is more likely to contain something you had not thought of.

D. Score them separately in two columns, since length and correctness are different properties to be weighed.

11 2. Testing a tool yourself

An arena leaderboard offers a style-controlled view alongside its headline ranking. Why is that the one to read?

A. It weights votes by the expertise of the voter, which removes casual judgements.
B. It restricts the comparison to prompts resembling professional work.
C. It statistically removes length and formatting effects, and the ordering changes when it does.
D. It excludes models that have been fine-tuned specifically to perform well in head-to-head comparisons.

12 2. Testing a tool yourself

One of your twelve prompts is code in a language you do not read. What should you do with its score?

A. Give it a 2 if it looks well organised, since structure is a reasonable proxy.
B. Average the other eleven scores and assign that, so the prompt does not distort the total.
C. Score it on whether the tool explained its reasoning, which is the part a non-specialist can legitimately judge.
D. Mark it unscored and have somebody qualified look at three answers, or replace the prompt.

13 2. Testing a tool yourself

A tool fails one of your prompts and you rewrite the prompt so it passes. What have you done?

A. Stopped the file testing anything, unless you note the rewrite and reset the history.
B. Improved the test, since a prompt a tool fails is by definition badly written.
C. Nothing significant, because the file measures tools rather than prompts.
D. Introduced a bias towards that tool, which is corrected by running the new prompt on every candidate again.

14 2. Testing a tool yourself

You set temperature to zero on an API and still get a slightly different answer on a second run. Why?

A. Temperature zero disables sampling only for the first token of the reply.
B. Providers run across large hardware fleets where floating-point operation order varies.
C. The provider silently raises the temperature when a request repeats, to avoid serving cached text.
D. Zero is interpreted as a request for the provider's default, which differs between models and endpoints.

15 3. The routes in

Why does moving a script between two providers, or to a local model, often take two lines?

A. Every provider publishes an adapter library that translates between formats.
B. Models are interchangeable at the weight level, so any client can load any of them.
C. One request format became a de facto standard, and most providers and local runners accept it.
D. Routers rewrite requests between formats automatically, which is what makes them worth the margin they charge.

16 3. The routes in

Why can a key never be safely embedded in a mobile app or in a web page's JavaScript?

A. App stores scan submissions and reject any package containing a credential.
B. Mobile platforms strip credentials from network requests as a security measure.
C. Keys are bound to a server's address at issue time and will not authenticate from a consumer device.
D. Anyone can read what their own device received, however the key is obfuscated.

17 3. The routes in

You are experimenting with a client's contract and want to use a provider's free developer credits. What is the problem?

A. Free tiers frequently permit the provider to use your inputs to improve the service.

B. Free credits usually expire within thirty to ninety days, so the work may be interrupted.

C. Free credits are capped at a context length too small for a contract.

D. Free tiers are rate-limited so heavily that a long document cannot be processed in one session.

18 3. The routes in

A phone has almost no free storage and the assistant's app is not listed in its app store. What is the route?

A. Install a lighter third-party client that wraps the same service.

B. Open the website and add it to the home screen from the browser.

C. Use the service over SMS, which is the only alternative to a native app.

D. Clear the phone's cached data until the official application fits, since the web version lacks most features.

19 3. The routes in

On a bundle-metered connection, which of these deserves the most care?

A. Long text conversations, because the whole history is resent each turn.

B. Downloading a local model, because it is the largest single transfer you will make.

C. Photographs and voice mode, because one photo costs about as much as a fortnight of text.

D. Thinking mode, because hidden reasoning tokens travel over the connection along with the visible answer.

20 3. The routes in

What single habit most protects you when letting a terminal agent edit your project?

A. Running it with the confirmation prompts disabled, so it completes work in one pass rather than many.

B. Asking it to produce the whole change at once, so you can read it as a single coherent piece.

C. Restricting it to a copy of the project, then merging the result once you are satisfied.

D. Committing before you start, so a diff is your review and a checkout is your undo.

21 3. The routes in

Why do streaming replies interact badly with a screen reader, and what usually helps?

A. Text appears token by token and may be repeated or interrupted; turning streaming off helps.

B. The reader cannot pronounce partial words; slowing the speech rate fixes it.

C. Screen readers ignore dynamically inserted text entirely; refreshing the page helps.

D. The reader announces the model's name before each fragment; disabling notifications in the product removes it.

22 4. What it actually costs

Roughly how much English prose is a thousand tokens?

A. About two hundred words, or half a page.

B. About seven hundred and fifty words, or a page and a half.

C. About two thousand words, or four pages.

D. It cannot be estimated, because tokenisation depends entirely on the specific model being used.

23 4. What it actually costs

A model with a two-hundred-thousand-token window is filled with a document, and you ask twenty questions about it in one conversation. What is the cost shape?

A. Twenty times one question, since the document is uploaded once.

B. Roughly the cost of one question, because the window is charged when it is filled.

C. The whole window is billed as input on every request, so it is twenty times the document.

D. Unpredictable, because providers apply automatic caching at different rates depending on load.

24 4. What it actually costs

You send fifty thousand items through a provider's batch route. What are the two practical consequences?

A. Around half price, and a guaranteed completion time you can put in front of a deadline.

B. Around a tenth of the price, and a limit of one batch per account per day.

C. Around half price, and a requirement that every request in the batch uses the same model and the same prompt.

D. Around half price, and results returned unordered so each request needs your own identifier.

25 4. What it actually costs

You want to cut an API bill. In what order should you act?

A. Send less and ask for less, then cache and batch, then route by difficulty, then accept worse output.

B. Switch to a cheaper model first, then optimise the prompt if quality suffers.

C. Negotiate a volume discount, then batch, then reduce the number of requests.

D. Turn off thinking mode, then reduce the number of users with access, then move the whole workload to a local model.

26 4. What it actually costs

A team of twelve has an AI seat licence and four people use it. What is the likely position?

A. The plan will downgrade unused seats automatically at renewal.

B. The organisation is paying three times what it needs, because seats are charged whether used or not.

C. The cost is justified, because unused seats keep the per-seat price lower for everyone.

D. The eight unused seats are harmless, since consumption pricing means an unused seat generates no charge.

27 4. What it actually costs

A subscription is advertised at twenty dollars. What can make the amount leaving your bank noticeably larger?

A. Currency conversion, which providers apply at a penalty rate set in their terms.

B. The provider's regional pricing, which raises the price in lower-income countries.

C. A foreign transaction fee, digital services tax, and in some countries a levy on outbound payments.

D. The payment processor's charge, which is added to the invoice separately and is typically around ten per cent.

28 4. What it actually costs

When is a cascade the wrong design?

A. When the small model is only slightly cheaper than the large one.

B. When the task involves more than one language, since the escalation signal becomes unreliable.

C. When the volume is very large, because the escalation step multiplies the number of requests you must manage.

D. When a person is waiting, because two model calls take twice as long.

29 5. Open models and your own hardware

You are building a product to sell and two open models are equally good for the task. Which licence removes a whole class of question?

A. Apache 2.0 or MIT.

B. A community licence with an acceptable-use policy, because it explicitly permits commercial use.

C. A non-commercial licence with a paid commercial option available from the maker.

D. Whichever licence the largest number of other companies have already built products on.

30 5. Open models and your own hardware

Two people disagree about whether a widely used model is open source. What does this course say to do?

A. Follow the Open Source Initiative's definition, which settles the matter for every model.

B. Read the actual licence rather than the word used to describe it.

C. Treat open weights and open source as interchangeable, since the practical freedoms are the same.

D. Check whether the training data is published, which is the single test that decides the question conclusively.

31 5. Open models and your own hardware

You are running a model on a Mac with Apple Silicon. Which format is worth preferring, and why?

A. GPTQ, because it is the fastest four-bit format on any hardware.

B. Safetensors loaded directly in Python, because it avoids a conversion step.

C. MLX, because it is built for Apple Silicon and is noticeably faster than GGUF there.

D. AWQ, because it is designed for GPU serving and unified memory behaves like a graphics card.

32 5. Open models and your own hardware

A model nearly fits in your graphics card's memory but not quite. What happens?

A. It runs a little slower, in proportion to the fraction that did not fit.
B. It refuses to load, since partial residency is not supported.
C. It quantises the overflowing layers automatically, trading a small amount of quality for the fit.
D. It spills into system memory and slows by an order of magnitude.

33 5. Open models and your own hardware

What is the characteristic sign that you have quantised a model too far?

A. Instruction-following slips: parts of the request are ignored or the format drifts mid-answer.
B. Facts become wrong while the writing stays fluent.
C. The model becomes slower, because lower precision requires more passes.
D. The model refuses more often, because compression damages the safety training before anything else.

34 5. Open models and your own hardware

Which of these is the clearest case where running a model yourself is the wrong answer?

A. A clinic that must keep patient notes on its own premises.
B. A second person now depends on it, so you have become the operator of a service.
C. A five-year archive of email that has to be classified once.
D. A lawyer under privilege who needs the same behaviour from a model in five years' time.

35 5. Open models and your own hardware

A fine-tuned community build promises better performance on your domain. What should you be careful about?

A. Fine-tunes cannot be quantised, so they need more memory than the base model.
B. Fine-tuned models carry a different licence from the model they were built on.
C. A fine-tune can lose capability elsewhere while gaining it on its target, and the card rarely measures that.
D. Community builds are usually converted from the base model months after release, so they lag behind on knowledge.

36 6. Data, terms and the law

Across the major providers, which route generally has the strictest default data terms?

A. The consumer free tier, because regulators scrutinise it most closely.
B. The consumer paid tier, because paying opts you out automatically.
C. The mobile application, because app store policies impose additional privacy requirements on developers.
D. The API, where inputs are generally not used for training by default.

37 6. Data, terms and the law

What is the most informative thing on a provider's sub-processor list?

A. Whether another model provider appears, and where each sub-processor operates.
B. The number of sub-processors, since fewer means a simpler data path.
C. Whether the list carries a certification badge from an external auditor.
D. The date the list was last updated, which is the only part a provider is contractually obliged to keep current.

38 6. Data, terms and the law

A vendor holds SOC 2 Type II and ISO 27001. What have you learned?

A. That the product's answers meet a documented accuracy standard.

B. That an auditor checked the company follows the security processes it says it follows.

C. That your data cannot leak from their systems.

D. That the model has been independently assessed for bias and for its behaviour on sensitive categories of question.

39 6. Data, terms and the law

You delete a conversation containing material you should not have sent. What has that achieved?

A. Nothing at all, since deletion in these products is cosmetic and removes only what your account displays.

B. Complete removal, including from any training data the provider has collected.

C. It limits further exposure going forward; it cannot remove anything already used in a completed training run.

D. Removal from the provider's live systems but not from its backup copies, which are purged on a published schedule.

40 6. Data, terms and the law

You remove the name column from a dataset of several thousand records. Is it anonymised?

A. Yes, provided no other column contains a name or a contact detail.

B. Yes, because identification requires a direct identifier by definition.

C. No, but it becomes anonymised once the records are aggregated into groups of at least ten.

D. No — a small number of ordinary attributes is often enough to identify a specific person.

41 6. Data, terms and the law

A provider's terms assign you their rights in whatever the model produces. Which question does that settle?

A. Whether the provider will claim ownership of it.
B. Whether the output is protected by copyright in your country.
C. Whether the output might infringe somebody else's rights.
D. Whether you may use the output commercially in jurisdictions where machine-generated work is unprotected.

42 6. Data, terms and the law

Which sentence belongs in a workable one-page AI policy for a small team?

A. We verify AI output before it is used.
B. Client-facing text is read by the person sending it, before sending.
C. Staff should exercise appropriate judgement when using AI tools.
D. All AI-assisted work must be reviewed in accordance with the organisation's existing quality assurance framework.

43 7. Beyond the chat box

Which single change most improves a beginner's results with an image tool?

A. Writing longer and more detailed prompts.
B. Generating at the highest resolution the tool offers.
C. Supplying a reference image of the product, room or person.
D. Using a paid hosted service rather than an open model on your own machine.

44 7. Beyond the chat box

What most affects the accuracy of a transcript?

A. Which service you pay for, since commercial engines are trained on far more audio.
B. The length of the recording, because quality degrades over long inputs.
C. Whether the transcript is requested as plain text or as a subtitle file with timings.
D. Audio quality and overlapping speech, before any choice of tool.

45 7. Beyond the chat box

Beyond obtaining consent, what is the practical household defence against voice cloning fraud?

A. A shared phrase or a callback rule for anything involving money or urgency.
B. Keeping recordings of your family's voices off public platforms.
C. Using a service that watermarks synthetic audio.
D. Answering unknown numbers only on a video call, so the caller's face can be seen alongside the voice.

46 7. Beyond the chat box

Which check catches most table extraction errors in one line of arithmetic?

A. Comparing the number of columns found against the number in the header.
B. Checking whether the extracted rows still add up to the printed total.
C. Running the extraction twice and comparing the two results.
D. Asking the model to state which rows it was least confident about, then checking only those.

47 7. Beyond the chat box

An assistant inside a spreadsheet can give you a formula or paste a computed value. Which should you prefer, and why?

A. The value, because it cannot break when the data changes.
B. Either, since the sheet looks the same and the answer is identical.
C. The formula, because it is auditable and recalculates when the data changes.
D. The value for reports and the formula for working files, because readers should not see the calculation.

48 7. Beyond the chat box

An organisation is deploying an office suite assistant. What should be done first?

A. Train everyone, so adoption is uniform from the first day.
B. Disable the assistant's access to email until a policy is agreed.
C. Benchmark it against a consumer assistant on the same tasks, to confirm the model is at least as capable.
D. Tidy file permissions, because the assistant surfaces what over-broad sharing made reachable.

49 7. Beyond the chat box

Which arrangement is the central unsolved security problem with agents today?

A. An agent that reads untrusted text while holding permission to act.
B. An agent that runs for many steps without a person watching it.
C. An agent that uses a model from a different vendor than the platform it runs on.
D. An agent whose logs are held by the platform rather than by the organisation operating it.

50 8. Living with it

At roughly what point does a saved prompt deserve to become a script?

A. When it is longer than a page, since long prompts are error-prone to paste.
B. When you are doing the same thing about twenty times.
C. As soon as it works, because a script is more reliable than pasting.
D. When more than one person needs it, because a script can be shared while a prompt cannot.

51 8. Living with it

Why put your stable reference material first and the specific task last in a long request?

A. It is a convention that makes prompts easier for colleagues to read.
B. Models process instructions before documents regardless of order, so this is purely cosmetic.
C. The request lands where attention is most reliable, and prefix caching matches the unchanging beginning.
D. Reference material placed first is treated as more authoritative than anything that follows it in the request.

52 8. Living with it

What is the honest response to the fact that a product's memory of you does not transfer to a competitor?

A. Rebuild it deliberately by re-telling the new tool everything over the first month.
B. Treat it as genuine lock-in and weight it heavily when deciding whether to switch.
C. Export the conversation history and upload it to the new tool, which reconstructs most of the same context.
D. Demote it: read the memory page, write the handful of durable facts into your own instruction file.

53 8. Living with it

You are the first person in your organisation using these tools well. What is the most useful thing to leave behind?

A. A shared folder of working prompts, each with a line saying what it is for.
B. A presentation showing what the tools can do.
C. A list of the tools you evaluated and the scores you gave them.
D. A training session for everyone, hands on, using real work rather than demonstrations of features.

54 8. Living with it

A team measures the effect of a new tool and reports an average saving per person. What is wrong with that number?

A. Averages are unreliable on samples smaller than thirty people.

B. The real finding is a distribution: transformative for a few roles, modest for most, negative for one or two.

C. Self-reported savings in time are always exaggerated by the people reporting them, and should be discounted by half.

D. An average taken over only two weeks is far too short a window to capture the learning curve properly.

55 8. Living with it

Which comparison makes a measurement of benefit flattering in a way that is easy to miss?

A. Comparing against a colleague who does the same task without the tool.

B. Comparing against the same task done six months earlier.

C. Comparing against doing the task perfectly by hand with unlimited time.

D. Comparing against the published figures from a vendor's own case study of a similar organisation.

56 8. Living with it

Which of these has held for years and is least likely to be out of date when you reread this course?

A. The ratio between a small model's price and a large one's.

B. What a sixteen-gigabyte laptop can usefully run.

C. Which company currently leads on long-document work, which has been stable for several release cycles.

D. That fluency is not a reliability signal.

Answers

Each entry gives the correct option, then what each wrong one reveals about how somebody was thinking. The second part is worth more than the first: a wrong answer here is a named misreading, not a mistake.

Lesson checks

1. What you are actually choosing — A

`B` — Assumes that different output means different weights, when the same weights given different input behave completely differently.

`C` — Prompt wording does help, but no phrasing can recover text the app never sent to the model.

`D` — Regenerating changes wording, not whether the document reached the model at all.

2. Reading a model's name — C

`A` — Confuses stability of behaviour with permanence of hosting; a snapshot promises the same weights while it is served, not that it is served forever.

`B` — Inverts the trade-off: the alias keeps working precisely by changing the weights underneath, which is the failure the pin was chosen to prevent.

`D` — Jumps to a conclusion the evidence does not support; deprecation dates for pinned builds are published months ahead and an expiry is not an early withdrawal.

3. Small, medium, large: what the size label buys — B

`A` — Treats every failure as a capability ceiling; dropped rules in a dense paragraph are usually a prompt-format problem and cost twenty times more to solve by upgrading.

`C` — Solves it by multiplying cost sevenfold when a formatting change usually fixes it, and it forfeits the model's ability to weigh rules against each other.

`D` — Takes a general rule as a guarantee; being the right class of task makes the small model a good candidate, not a verified one, and twenty outputs still have to be read.

4. Thinking modes, and what you pay for them — A

B — Names a plausible-sounding boundary that is not the actual one; the limit is knowledge versus structure, not subject area.

C — Assumes more of the same process fixes it; the improvement curve flattens early and no amount of deliberation retrieves a fact the weights do not contain.

D — Misapplies a true caveat about reasoning summaries; the answer itself was wrong here, which no amount of paraphrasing explains.

5. The context window is a ceiling, not a promise — A

B — Reaches for the model layer when the evidence points at the product layer; a weaker model would degrade on the keyword question too, and it answered that one perfectly.

C — Treats the advertised window as the operative cause; window size sets what is permitted, while the product's decision about what to send decides what happens.

D — Invents an explanation that does not fit the observation, since a guessed contradiction would not have named two specific pages that genuinely conflict.

6. The instructions you never see — A

B — Compares one wrapper with another wrapper; both carry their own instruction blocks, so agreement between them is not evidence about the weights.

C — Establishes what is against the rules, not where the rule is enforced, and product wrappers commonly refuse well beyond the published policy.

D — Trusts an output the model frequently reconstructs rather than retrieves, producing a convincing but invented account of its own configuration.

7. What 'it can browse the web' actually means — B

A — Blames a file-permission model that is not the operative constraint; sandboxes generally allow writing to their own temporary storage.

C — Imagines an allowlist of operations; the sandbox runs arbitrary code, which is exactly why it is isolated from the network.

D — Confuses two tools: search returns page extracts as text and cannot supply a library to the sandbox or hand it live data to compute over.

8. 'It understands images' — what that buys — B

A — Assumes a specialised model removes the constraint; the loss happens when pixels are downsampled into a token budget, which applies to specialised vision models too.

C — Requests a self-report the model cannot ground, since it has no signal distinguishing a digit it read from a digit it reconstructed.

D — Adds context to a problem of resolution; more pages would consume more of the same token budget and make small print less legible, not more.

9. When the engine gets swapped — A

B — Applies more compute to a task whose failure is a changed interpretation of the prompt, and classification into known buckets rarely benefits from deliberation.

C — Buys time by freezing behaviour, but pinned snapshots are themselves deprecated on a published date, so it postpones the migration rather than resolving it.

D — Treats an average improvement as a defect; the provider shipped a better model overall and will not roll it back for one prompt's assumptions.

10. Where they differ, and where it is marketing — C

A — Objective-looking is not the same as relevant; a measurement only helps if the test resembles the job.

B — Test-tuning is real, but inverting the leaderboard is superstition pointed the other way.

D — They change because the models change; the problem here is fit to his task, not instability.

11. The file you will keep for years — B

A — Assumes fairness is only about consistency; criteria derived from one tool's output describe that tool, so applying them evenly still measures how closely rivals imitate it.

C — Treats an anchoring problem as an order effect; rotation helps with fatigue and ordering, but the criteria themselves stay contaminated.

D — Adds a reviewer to a standard that is already shaped by the first tool's output, so a second person applies the same contaminated criteria.

12. The prompts where you already know the answer — B

A — Attributes to missing knowledge what the first correct answer already disproves; the fact was retrieved successfully before the challenge arrived.

C — Invokes a long-context failure at a length where nothing has been lost; two short turns sit well inside any window.

D — Assumes a mode change that did not occur, and deliberation triggered by doubt would not systematically favour the doubter.

13. Testing the language you actually work in — A

B — Draws a quality conclusion from a counting fact; accuracy is driven by how much of the language was in training, not by how finely the text is chopped.

C — Reverses cause and effect: token counts come from how the tokeniser was fitted, and nothing about the count implies an internal translation step.

D — Assumes a billing model that does not hold on any major API, where input and output tokens are exactly what is metered.

14. Taking the brand off the answers — C

A — Reads a two-point gap on twelve items as a real ordering, when that margin is inside what scoring noise on a sample this small produces.

B — Chases a distinction that would not change the decision; more items would sharpen a gap too small to act on either way.

D — Reintroduces exactly the bias the blinding removed, and a seven-point deficit is far beyond what style preference explains.

15. Run it five times — A

B — Mistakes a knowledge gap for a sampling setting; temperature zero would return one consistent date, with nothing making it the right one.

C — Explains away identical prompts as different ones; isolation is what makes the three draws independent and therefore informative.

D — Assumes a retrieval step that may not have run at all, and cited sources would be visible and checkable rather than silently contradictory.

16. A rubric you can apply in ten seconds — D

A — Values a house style set by a system prompt as if it were substance, when the same content is present in both.

B — Declares the difference irrelevant when the reading cost is real and falls on you every day the tool is used.

C — Reads unrequested structure as instruction-following, when the instruction came from the product rather than from you.

17. Reading a leaderboard without being fooled — B

A — Defends the default as more reliable when the adjustment exists precisely because the default conflates presentation with substance.

C — Borrows a static-benchmark failure; arena prompts are supplied live by voters, so the problem there is preference, not memorisation.

D — Invents a version difference; both views are computed from the same votes over the same models, differing only in the statistical adjustment applied.

18. Testing the edges: refusals and house manners — C

A — Treats an unverified sentence as a credential check; the product has no way to confirm the role and would respond the same to anyone typing it.

B — Calls a deliberate default a defect; caution before context is a design choice, and calling it a bug obscures that the model responds to stated context.

D — Invents a prohibition on describing your own situation, when providers generally expect users to supply the context their questions need.

19. Making the decision, and recording it — A

B — Assumes the record expires quickly, when the model strings and scores are exactly what makes a later change detectable.

C — Borrows the language of experiments for a decision aid; twelve prompts scored by one person is not made rigorous by writing a condition down.

D — Casts the record as an argument to win rather than a trigger to watch for, which is the use that actually saves time.

20. The web app, the phone app and the desktop app — A

B — Describes ordinary end-of-term billing, which would stop after one cycle rather than continuing indefinitely.

C — Invents a second purchase; one payment through the store fully explains the charges without a duplicate.

D — Overstates the constraint: store subscriptions are user-cancellable, just from the store rather than from inside the app.

21. Your first API call — C

A — Attributes to local execution a limit that is about the interface, since open reasoning models do run locally and expose their own controls.

B — Invents an authentication failure that would have returned an error rather than a working answer with one setting missing.

D — Places the behaviour in the client library, when it is the receiving endpoint that ignores fields it does not recognise.

22. Keys, spend caps, and the bill nobody wants — D

A — Describes a rescue mechanism rather than the problem; scanning is inconsistent and the exposure exists whether or not anyone flagged it.

B — Points at stale clones, a side issue next to the fact that the secret sits permanently in the public history.

C — Claims an irreversibility that does not exist; revoking a key in the console is immediate and is exactly the right response.

23. Building your own chat app for nothing — B

A — Asserts an encryption difference that does not follow from licensing; either kind may or may not encrypt local storage.

C — Confuses open with local; an open front-end still sends every request to whichever provider you configured.

D — Invents a store requirement; some clients proxy requests and others do not, which is exactly why it has to be checked.

24. Routers: one key, many models — C

A — Assumes rolling updates to released open weights, which are published as fixed versioned files rather than changed underneath users.

B — Confuses fallback between models with routing between hosts of the same model; a model substitution would normally be reported in the response.

D — Attributes to sampling a difference in the stated context limit, which no temperature setting can change.

25. Where a coding assistant lives — C

A — Names a real inconvenience of large changes but not the reason this one is risky, since merge conflicts surface loudly rather than silently.

B — Assumes quality degrades with breadth; the problem is your ability to review the result, not the model's ability to produce it.

D — States a true property of commits as the main harm, when the deeper issue is that the change was never examined.

26. Extensions, wrappers and the permission dialog — C

A — Blames a platform change for content the extension is inserting, which a broken cleaner would not add on its own.

B — Explains away a pattern that appears specifically after an update to this extension, rather than across unrelated installs.

D — Assumes a disclosed change; ownership transfers followed by silent monetisation are the documented pattern for this category.

27. When the phone is the computer — C

A — Ignores an order-of-magnitude gap; overhead is measured in kilobytes while an unresized photograph is measured in megabytes.

B — Locates the cost in the reply, when replies are text and remain small regardless of what prompted them.

D — Assumes automatic compression that browsers do not perform; resizing has to be done in the gallery or the app.

28. Voice, screen readers and hands-free use — B

A — Shortens the window in which the problem occurs without removing it, and makes the model choice serve the interface rather than the work.

C — Trades a readable, reviewable answer for speech, which is worse for anything the user needs to keep or check.

D — Reduces the symptom in proportion to length without addressing the announcement behaviour that causes it.

29. The free tiers, and their traps — C

A — Heavy load shows up as slowness or errors, not as a quiet drop in reasoning quality.

B — Long threads do lose earlier detail, but that produces inconsistency inside one thread, not weaker answers in new ones.

D — Updates happen, but they roll out over days and would not land exactly when his usage piled up.

30. When paying is worth it — B

A — They overlap heavily, but house style, tool access and language coverage still differ in ways she can feel.

C — Memory features store notes about you; they do not raise a model's capability on tasks another model handled better.

D — Per-token is cheaper for light use and far more expensive for heavy use — the error is the word 'always'.

31. The unit you are billed in — B

A — Would make the totals equal by coincidence, but 30,400 tokens against 6,900 is a fourfold difference in count.

C — Assumes per-request pricing that no major API uses, where input and output tokens are metered separately.

D — Imagines a compression discount; long inputs are billed in full, and caching only helps when the same text is sent again.

32. Costing a job before you run it — C

A — Invokes demand pricing that these APIs do not use; published per-token rates do not vary with load.

B — Points at a trend that has generally moved the other way, and a nine-fold jump is far beyond any price revision.

D — Invents a batch-position effect; each request is independent and knows nothing about its place in a queue.

33. The two discounts almost nobody uses — D

A — Invents a low size cap; long stable preambles are exactly what caching is designed for, with minimums rather than tight maximums.

B — Adds a role restriction that does not exist, when the match is on the leading text regardless of which role carries it.

C — Attributes to a content rule what is a plain string comparison; the cache has no notion of what a date means.

34. Spending money where it matters — D

A — Dismisses a well-documented pattern as randomness; the group persists at low temperature because large models genuinely overthink simple cases.

B — Assumes strict dominance that does not hold, and prompt tuning does not remove a group that arises from the models differing in kind.

C — Attributes to sample size a difference that grows rather than vanishes with more items.

35. Subscriptions, credits and pay-as-you-go — C

A — Prescribes a route that fits automation rather than human chat use, and does not address seats being billed for absent users.

B — Assumes pooled allowances that seat pricing does not generally provide; the charge follows the seat, not the usage.

D — Confuses the plan's features with per-seat charges; the features come with the plan, and the empty seats add cost without adding anything.

36. The costs that are not on the invoice — C

A — Sorts tasks by difficulty when the deciding property is checkability; hard code that either runs or does not is cheap to verify.

B — Credits prompt skill, which changes answer quality without changing how long verification takes.

D — Restates a sound guideline rather than the mechanism, and drafting a claim you cannot verify is still expensive.

37. Paying for it from where you live — B

A — Invents a surcharge; the common pattern is a single dollar price worldwide with local taxes and fees added on top.

C — Predicts a one-off that would disappear next month, when transaction fees and tax recur every cycle.

D — Attributes a consistent 25% gap to rate movement, which is normally a fraction of a per cent over a few days.

38. A budget that actually holds — A

B — Understates enforced caps, which do stop service on major providers rather than merely warning.

C — Claims a price advantage that does not generally exist; prepaid and invoiced use are billed at the same per-token rates.

D — Draws the wrong boundary: a leaked key can spend whatever credit is loaded, and the balance is the limit rather than the card.

39. The open models — D

A — Downloadable is not the same as open source; several widely used licences carry real conditions.

B — Almost no released model publishes its training data, and its absence does not block deployment.

C — Offline capability comes from running weights yourself; it is not a licence term.

40. Running a model on your own machine — A

B — 4-bit costs a little quality but works fine; the failure here is capacity, not precision.

C — CPU inference works, just slowly. Model size, not processor type, is what stops her.

D — 4-bit is roughly half a byte per parameter, so 70B lands near 40 GB — the file size is right.

41. Working out what your machine will run — D

A — Invokes a fallback mechanism where the observed symptom is memory pressure, and exceeding the window normally truncates rather than slows.

B — Invents dynamic precision; quantised weights are fixed on disk and do not change with input length.

C — Describes a retrieval pipeline that a plain local runner does not perform on pasted text.

42. Reading a model card before you download — C

A — Claims a size advantage; the format changes how tensors are stored and loaded, not how much space the same weights occupy.

B — Attributes metadata guarantees to a format that stores tensors, with provenance still coming from the repository.

D — Confuses container format with quantisation, which can be applied to weights stored either way.

43. Choosing a quantisation — A

B — Treats parameter count as decisive while ignoring that aggressive quantisation removes much of what those parameters encoded.

C — Asserts an impossibility that 2-bit quantisation specifically overcomes, which is why the choice arises at all.

D — Splits the decision plausibly, but low-bit quantisation harms recall as well, so the split does not hold up in testing.

44. Renting an open model instead of running it — D

A — Assumes a standard that does not exist; providers independently choose precision and many do not disclose it.

B — Jumps to a conclusion and offers a check that will not settle it, since sampling variation swamps quantisation differences in a few outputs.

C — Confuses who pays for memory with who feels the quality effect, which lands entirely on your answers.

45. The licence is a business decision — A

B — Treats a definitional dispute as a legal prohibition, when these licences do permit commercial use on stated conditions.

C — Generalises from the user threshold to the whole licence, ignoring attribution, downstream terms and acceptable-use clauses that bind everyone.

D — Waits on a response these companies do not generally provide to individual users, while the published terms already answer the question.

46. When running it yourself is the wrong answer — A

B — Rests on a capability claim that does not hold, as a 70B on consumer hardware still trails the frontier on hard reasoning.

C — Counts other uses as savings against this decision, when they would be reasons to buy the card regardless of the subscription.

D — Elevates running cost above the purchase price, when the card itself is the dominant number in this comparison.

47. A working setup with two models — D

A — Points at a missing feature when the real problem is that your own difficulty estimate is unreliable in the moment.

B — Invents an accuracy advantage; local models are chosen here for where the computation happens, not for better handling of the content.

C — Assumes a correlation between sensitivity and simplicity that does not exist; contracts are both confidential and hard.

48. Who reads what you type — B

A — Intuitive, but several products have trained on paid consumer conversations by default; business tiers and APIs are what reliably change it.

C — They are usually separate settings, and turning off history may not stop training.

D — Data already absorbed into a training run cannot be pulled back out of a trained model.

49. Reading the terms in ten minutes — B

A — Reads a narrow promise as a general one; training and retention are separate practices with separate statements.

C — Picks a genuine but minor ambiguity while missing the storage, access and jurisdiction questions the sentence never touches.

D — Raises a real limit of past training runs, which is not what this forward-looking sentence is silent about.

50. What a business agreement actually buys — A

B — Notes a genuine limitation of audit timing while accepting the false premise that a current report would establish answer quality.

C — Splits audit scopes as if that were the gap, when no audit of either kind speaks to whether an answer is correct.

D — Correctly limits the scope to the provider, then still treats the report as bearing on output reliability.

51. Your country, your rules — B

A — Checks organisational discipline, which is evidence for a decision rather than the decision, and says nothing about lawful basis or transfer.

C — Raises a real clinical-safety question that sits outside data protection law and does not answer whether the sending is permitted at all.

D — Treats a general consent as sufficient, when purpose limitation means consent to one processing does not extend to another.

52. Who owns what it produces — B

A — Points at a licence-back term that limits the provider's use, not the designer's ability to enforce against a third party.

C — Introduces a real parallel regime while sidestepping the authorship problem, and trademark protection depends on use and registration rather than replacing it.

D — Describes a practical risk of reproducibility rather than the legal reason, since originality does not turn on whether a process could repeat.

53. Redaction that actually works — C

A — Describes one specific mechanism rather than the general problem, and the text remains extractable whether or not annotations can be hidden.

B — Imagines a visual artefact, when the failure is that the characters themselves were never removed from the file.

D — Relocates the text to metadata; it stays in the page's own content stream, which is why plain extraction recovers it.

54. Secrets, and what not to paste — B

A — Focuses on where a copy is visible, when the decisive fact is that the key itself is now usable by anyone who has seen it.

C — Treats a processing delay as the issue; even instant deletion would leave the key compromised.

D — Assumes single conversations enter the weights, which is not how training works and misses the immediate risk entirely.

55. Accounts, sharing and devices — D

A — Names a real inconvenience while missing that the history and memory are personal correspondence rather than a resource pool.

B — Focuses on enforcement risk rather than on what the account actually exposes to the people sharing it.

C — Invents a technical limit; conversations can be deleted individually, and the exposure exists long before anyone tries.

56. What your workplace probably has not decided — A

B — Argues from competitiveness, which is a business case rather than the risk mechanism the ban actually creates.

C — Invents a legal exposure; internal policies are ordinarily enforceable as employment terms.

D — Describes a consequence of not buying rather than the reason the ban backfires, since nothing stops a banning organisation from negotiating.

57. Choosing an image tool — B

A — Optimises for appearance while ignoring that misrendered brand text makes an otherwise beautiful image unusable.

C — Prioritises throughput, which matters once the output is usable at all but not before.

D — Applies a confidentiality rule where the decisive issues are legible text and legal cover for commercial output.

58. Transcription and translation — B

A — Assumes a specialisation that does not overcome poor audio, where the information simply is not present in the recording.

C — Names a real but narrow failure that produces obviously wrong output rather than scattered errors through a mostly correct transcript.

D — Repairs plausible-looking text after the fact, which can smooth errors into confident wrong sentences rather than recovering what was said.

59. Voice and speech tools — D

A — Protects the outputs while leaving the underlying capability — a voice model of a named executive — unaddressed.

B — Raises a presentational concern that matters less than establishing consent and a way to verify genuine instructions.

C — Optimises for fidelity, which increases rather than reduces the risk that a fraudulent message will be believed.

60. Video tools, generated and edited — D

A — Names a real pricing detail that adds a fixed multiplier, rather than the variable discard rate that dominates the total.

B — Treats the clip limit as a hard blocker, when the standard method is assembling many short generations.

C — Adds post-production cost, which is real and small next to the number of generations thrown away.

61. Research and search tools — D

A — Grants authority on the basis of volume, when concentration on a single commercial domain is what needed explaining.

B — Explains why one site might rank often without addressing that a single commercial voice now carries the report.

C — Blames a counting artefact rather than the substantive problem that one interested party supplied nearly all the content.

62. Documents, PDFs and OCR — C

A — Detects sampling variation rather than systematic misreading, which repeats consistently across runs.

B — Improves input quality without providing any way to detect an error that still occurs.

D — Requests a self-report with no grounding, since the model cannot distinguish a digit it read from one it reconstructed.

63. Spreadsheets and data tools — A

B — Produces a description of the intended logic, which will sound correct while the type mismatch that broke the join goes unmentioned.

C — Would catch a wrong total in many cases, but a summary of the surviving rows can look entirely reasonable on its own.

D — Relies on noticing an absence, and a chart of the remaining groups looks complete when nothing marks what is gone.

64. Meeting notetakers — A

B — Names a genuine summary failure while missing that the material is exposed to the whole organisation.

C — Points at duration when the immediate exposure is access, which harms the moment it exists.

D — Identifies an accuracy issue that matters less than every colleague being able to read the conversation.

65. Office suite assistants — B

A — Assumes the index overrode access controls, when it returned only what each user was already permitted to open.

C — Describes a staleness problem, where the issue is current permissions that were always too wide.

D — Invents a provisioning mistake; the same outcome occurs with ordinary accounts and pre-existing sharing.

66. Agents and automation platforms — D

A — Buys a real improvement while leaving the compounding structure intact, so the same failure returns as soon as the sequence lengthens.

B — Assumes failures are random rather than systematic, and a confidently wrong observation is retried into the same wrong answer.

C — Ignores that the failed 40% arrive as plausible completed work that someone must detect and unpick.

67. Specialist and industry tools — A

B — Values phrasing, which a well-written prompt supplies to a general assistant at no cost.

C — Treats a guardrail as the value, when a narrower scope does not make the answers inside that scope any more checkable.

D — Mistakes positioning for capability; the workflow claim is common to every wrapper in the category.

68. From chatting to a workflow — D

A — Invents a priority ordering; examples work by showing the target rather than by outranking other instructions.

B — Overstates the mechanism into template-filling, when the model still adapts the shape to different material.

C — Reverses the usual lengths, and a full example commonly costs more tokens than a sentence describing it.

69. Keeping your work portable — C

A — Overstates prevalence and rigidity; annual plans are optional and their cost is one factor rather than a bar to leaving.

B — Treats history as unrecoverable when most products export it, and old conversations are rarely what stops a switch.

D — Names a real friction that fades within about a fortnight, which is short next to rebuilding a year of context.

70. Two tools, and a date in your calendar — D

A — They are not independent — both read overlapping web text and can repeat the same common error together.

B — Disagreement genuinely would have told her something; agreement is just weaker evidence than it feels.

C — Different models converge on widely-repeated facts constantly without sharing anything.

71. When your tool is withdrawn — C

A — Assumes a downgrade, when successors are typically stronger overall and the regression is specific to this prompt.

B — Predicts truncation, which usually produces obvious failures rather than subtly worse output, and windows have generally grown.

D — Prescribes starting over, when the tuning encoded real requirements that only need stating explicitly.

72. Introducing a tool to other people — D

A — Prescribes instruction in a general skill, when the gap is that no specific task had been identified as worth doing.

B — Focuses on which product, when the same collapse follows from a mandate regardless of the tool chosen.

C — Criticises the metric, which obscured the failure rather than causing it.

73. Measuring whether it actually helped — D

A — Notes a comparability problem while accepting that the count would mean something within a single role.

B — Refines the unit rather than questioning whether any count of interactions tracks work completed.

C — Raises a coverage gap, which would matter only if the metric were sound to begin with.

74. Keeping up without drowning — D

A — Prefers a source for its authority, when the value is that it covers the specific tools you depend on.

B — Narrows to one item; changelogs also carry pricing, behaviour and feature changes that matter well before a deprecation.

C — Values convenience of timing over the fact that these are the only changes with direct consequences for your work.

75. When the right answer is not to use it — C

A — Draws a category line that would rule out perfectly sound refusals based on tested, current inadequacy.

B — Confers legitimacy through paperwork, which records a position without testing whether it is well founded.

D — Uses general familiarity as a credential, when a refusal still needs evidence about this particular task.

76. What changes, and what does not — D

A — States a hardware-to-size relationship that shifts as models grow more efficient and memory becomes cheaper.

B — Quotes a price point, which is among the fastest-moving numbers in the whole field.

C — Describes a gap that has been narrowing steadily, and where the boundary sits is exactly what keeps moving.

Module test — 1. What a tool is made of

1. A

The product layer decides what the model is actually given. A product that retrieves splits your file into chunks, searches for the ones that look relevant, and sends those — which is excellent for 'what does it say about indemnity' and poor for 'summarise the whole thing', because the model never sees the whole thing. Same weights, worse question.

2. B

A pinned snapshot gives you the same weights next month as today. An alias is re-pointed whenever the provider ships, and your prompt was tuned to the specifics of the old build. Pinning trades a quiet drift for a loud, scheduled migration you control — and the error you eventually get when a snapshot is retired is a kindness, because the alias would have degraded silently instead.

3. C

Look at how the outputs are wrong before moving up. Wrong facts mean the model does not hold the knowledge and you move up. Ignored instructions usually mean several conditions were packed into a paragraph, and the middle ones are the ones dropped — a numbered list frequently makes the same small model pass, at a twentieth of the price.

4. D

Thinking modes help where a wrong step produces a contradiction the model can notice — arithmetic with units, constraint problems, debugging, planning. Recall has no internal structure to check, so longer deliberation cannot fetch a fact the model does not hold. It raises your confidence in an answer more reliably than it raises the answer's accuracy, and you pay for every hidden token at the output rate.

5. A

The failure has a name — lost in the middle — and it holds on nearly every model at every price. Placing the instruction at the end puts it where attention is most reliable, and it costs nothing. On a paid API that ordering also suits prefix caching, but caching is a billing mechanism and has no effect on how carefully anything is read.

6. B

The loop is: the model asks for a tool, the product runs it, and the result is pasted into the conversation as text. Garbage in the tool result is garbage in the answer, and the model has no independent way to know. A resolving link is not evidence either — models produce citations that look right and point at a page that does not say what the answer says, so click two of them.

Module test — 2. Testing a tool yourself

1. C

Pre-commitment is what separates a test from a vibe. Read a fluent answer first and your standards reorganise themselves around what impressed you, without your noticing. Two minutes per prompt spent writing the line beforehand is the difference, and it is the single habit worth taking from this module if you take only one.

2. D

Products sample from a probability distribution rather than always taking the most likely token, so a single run is a sample and not a measurement. Run the prompts that matter three times per tool: three failures from three is a problem, one from three may be an unlucky draw. The same logic disposes of the screenshot of a remarkable answer, which is one draw selected for being remarkable.

3. A

A tool that has been told for six months that you work in logistics looks better on a logistics prompt. A fair run uses a fresh or private window for each candidate, memory and custom instructions off, and the same context pasted into all three. Order effects are real and shuffling handles them; this confound is larger and shuffling does nothing about it. Twelve prompts by one person is a small sample that still beats a headline by an enormous margin.

4. B

Once everything at the top scores above ninety, the differences live in the handful of questions that are ambiguous or wrongly labelled, and several widely quoted benchmarks are known to contain errors in a few per cent of their answers. Add contamination and best-case reporting settings, and a gap of a point or two carries no information about your work.

5. C

A model reached raw through an API will discuss things the consumer app declines, because the refusal sits in the product's instructions rather than the weights. That is why false refusals — the expensive failure for professionals — vary so much between products running the same model, and why a capable model inside a cautious product can be less useful than a weaker one inside a product that trusts its users.

6. D

Without a change-my-mind list, every announcement is potentially decision-relevant, so you read all of them and act on some at random. With one, almost everything is filtered in three seconds. It also catches the quieter failure: having paid for something, people defend the choice and stop noticing the free tier caught up, and a condition written down while you were still neutral is the only reliable way to see it has been met.

Module test — 3. The routes in

1. A

It is the identical subscription with a store's commission added. The related trap is cancellation: a plan bought in an app store is cancelled in the store's subscription settings rather than in the app, and people who cancel in the app find they are still being charged. Buy it in a browser and cancel where you bought it.

2. B

The response tells you why generation stopped. A finish reason of stop means it ended naturally; length means it hit the cap. This is almost never the model failing — it is a maximum-output setting, either yours or a default somewhere in the front-end. Raise it, or ask for a shorter answer, and the truncation goes away.

3. C

Ignoring a file after it has been committed does nothing, and deleting it in a later commit does not remove it from the history. Automated scanners find keys on public repositories within minutes, and this is industrialised rather than rare. The only fix that works is to revoke the key in the console and issue a new one: twenty seconds, no cost, and there is no situation where waiting to be sure is the better move.

4. D

Every turn resends everything before it, so most of a long conversation's cost is earlier turns being re-read. Capping the resent history at around the last ten messages removes that with almost no effect on quality, and it is off by default in all of these applications. Shortening the output helps as well, but output is the smaller number when your inputs are long.

5. A

The same model name is served by several companies. One may run a four-bit build with a sixteen-thousand-token window while another serves eight-bit at a hundred and twenty-eight thousand, and the router chooses per request. Good routers let you pin the host or filter by quantisation and context length, and that setting is usually off by default.

6. B

That is what the line says, and most AI extensions request it because reading the current page is the feature. It is not automatically wrong; it means the extension deserves the scrutiny you would give a person you were handing your browser to. Check the publisher, the age and user count, the newest reviews — extensions get sold, and the classic pattern is an update that inserts tracking into a tool that already has permission to change every page — and the data-use disclosure.

Module test — 4. What it actually costs

1. C

Output usually costs three to five times input, which is why a comparison quoting one number is quoting the wrong one half the time. But the shape of your work decides which price matters: reading long things and answering briefly buys input, while generating long documents from short briefs buys output. Thinking tokens are billed as output and are real, and they can be turned off or given a budget.

2. D

A request that returned a bad answer was still billed, a three-attempt retry loop triples the worst cases, and runs get abandoned halfway and restarted. A third is a realistic allowance and anything under a fifth is optimism. Batch is a genuine discount, but it is a decision about latency rather than a correction to the estimate.

3. A

The cache matches on a byte-identical prefix, so one variable field near the top invalidates everything after it. Move the date and the name to the end, after the question, and the whole block above stays cached. Expiry is real — caching pays on bursts and does nothing for one request an hour — but it is not what is breaking this request.

4. B

Large models overthink simple, constrained cases, and that group is consistently bigger than people assume. Both-wrong points at the prompt or the task rather than the model. Large-right-and-small-wrong is exactly what you pay the large model for, and the question is whether your escalation rules catch it — if they do not, tighten the rules rather than abandon the design.

5. C

A draft produced in thirty seconds that takes twenty-five minutes to verify has cost you five minutes. Tasks that are cheap to check — code that runs, arithmetic you can recompute, a summary of a document in front of you — give nearly pure gain. Tasks that are expensive to check give the largest illusory returns, because the saving is visible and the risk is not.

6. D

Free tiers give a limited number of messages on the strong model and then switch you down, and the only sign is that answers get shallower. Look for the model name in the interface before forming an opinion. The allowance also resets on a rolling window rather than at midnight, which is why the heavy task belongs at the start of one.

Module test — 5. Open models and your own hardware

1. A

Open weights means you can download, run and usually modify the finished numbers. It rarely means the recipe. Almost no open-weights release publishes its training data, which is why you cannot check for contamination, assess bias from the source material, or answer a customer asking whether their data was in it. AI2's OLMo family is the notable exception that publishes both.

2. B

The KV cache grows with the amount of text in the context, and on a long document it can rival the size of the weights. The fixes are settings rather than hardware: lower the context setting, quantise the cache to eight-bit, and close the browser, which routinely holds several gigabytes. A bigger model is not the answer, and usually neither is a bigger machine.

3. C

The curve is not a straight line. Eight-bit is indistinguishable in practice, four-bit is the standard choice and fine for everyday work, three-bit starts to slip on instruction-following, and two-bit is usually a false economy. Importance-matrix builds are genuinely better at the same file size, particularly at the low end, and they do not turn two bits into four.

4. D

The older format uses a general object-serialisation scheme that executes code while loading, which makes a downloaded file a program as well as data. Safetensors stores plain arrays of numbers. For running on a laptop you then want a GGUF conversion as well, which is a separate question about the runner rather than about safety.

5. A

The same model name from two providers is not the same service. Precision, served context length and speed all differ — one may quietly serve a heavily quantised build at a lower price — and the data terms differ most of all. A cheap provider that retains your client's contract is not cheap, and if ten minutes cannot answer the retention, training and human-review question, that is your answer for anything sensitive.

6. B

Is there a client name in this? is checkable in a second. Is this hard? is a judgement that gets worse the busier you are, which is exactly when the decision gets made. Writing the rule down once — confidential to the local model, routine and bulk to a cheap hosted one, the hard remainder to the strong one — removes the choice from the moment you are least able to make it.

Module test — 6. Data, terms and the law

1. C

Training, retention and human review are three separate promises with three separate answers, and companies answer the first loudly and the other two quietly. A provider can truthfully say it does not train on your data while keeping conversations for thirty days and allowing staff to review flagged ones. Almost every policy contains that review sentence, which means the honest mental model is not that a machine reads this and forgets.

2. D

The well-formed version names the route: does this company train on what I send through the free web app, on a personal account, with the default settings. The API is commonly the strictest and the consumer free tier the loosest, and the answer often differs from whatever a headline about the company said. If ten minutes searching for train, retention, human review, third part, delete and transfer produces nothing clear, treat that as the answer.

3. A

A business tier buys a contract rather than a better model: a training commitment as a term rather than a toggle, a processing agreement, a sub-processor list, retention controls and sometimes regional processing. None of it makes the output correct, and none of it moves the responsibility — you are the one who decided to send the material, and the agreement is evidence of due diligence rather than a transfer of liability.

4. B

Using a model to help you decide is different from letting it decide. A person in the loop who genuinely reviews and can overrule, with a record that they did, is what separates the two. It matters most where the effect is significant — a rejected application, a refused benefit, a candidate screened out — and it is a provision to look up for your own jurisdiction before you build the process rather than after.

5. C

A rectangle hides text from the eye and not from extraction, and this has produced published court filings whose redacted names could be copied out with a mouse. Proper redaction removes the text. The same class of problem covers document metadata, hidden spreadsheet rows, image location data and screenshots, and checking afterwards with a text extractor takes ten seconds.

6. D

Rotating is the only step that genuinely fixes anything and it takes a minute. Deleting the conversation is worth doing and limits further exposure, but it does not undo storage that has already happened. Checking retention and writing down what happened come afterwards, and if the material was somebody else's, telling whoever is responsible matters too, because a reporting clock started when it happened rather than when you mentioned it.

Module test — 7. Beyond the chat box

1. A

If you can select and copy the text, it is there, and extracting it is exact, instant and free. Sending a text-layer file through a vision model is slower, more expensive, and capable of changing a digit where the correct characters were already sitting in the file. Recognition is for when there is no text layer; a vision model is for when the question is what the page means rather than what it says.

2. B

The characteristic failure is the silently dropped row — a filter or a join that removes records nobody counted, in code that runs without complaint. A row count makes it visible. The same reasoning is why you prefer a tool that shows you the code: a wrong column or a type mismatch on a join is obvious in three lines of code and invisible in the chart.

3. C

The direction of the failure changes. A search-grounded tool is bounded by what the search returned, so a commercial question answered from three marketing pages is fluent and worthless. A tool restricted to your own sources answers from material you chose and cites a passage you can check in a click. It is not perfect: retrieval can miss a passage worded differently, and the claim that the sources are silent is sometimes wrong.

4. D

Reliability compounds: ten steps at ninety-five per cent succeed about sixty per cent of the time, and twenty steps about thirty-six. Errors do not merely accumulate, they mislead — a wrong observation at step four sends the plan somewhere useless for the next six. The answer is not to avoid agents but to run three or four steps at a time, ending in a result a person or a rule can verify.

5. A

A meeting that ended undecided is summarised as a decision because summaries are written in the shape of conclusions, and nuance goes with it: 'we could do X, though Y worries me' becomes 'agreed to do X'. The same class of error attributes an action to the wrong person, which is consequential because people act on the list rather than the transcript. The defence is that the person who ran the meeting reads it first.

6. B

Nearly all the practical value is in editing, and the free path there is professional: Resolve's free version includes transcription, subtitle generation, voice isolation and reframing, with Shotcut, Kdenlive and CapCut covering the lighter end. Generation is priced per second with a high discard rate — five to fifteen attempts for a usable five-second shot — and works best as short ingredients cut into real footage.

Module test — 8. Living with it

1. C

An instance of an answer you liked communicates more about tone, length and register than a paragraph describing them, and it is the cheapest quality improvement available anywhere in this course. The rest — role, context, format, standards — is what you had to explain every time anyway. Keep the master copy in a plain text file of your own, not only inside one product's project feature.

2. D

Disagreement is the working alarm: when two models differ on a threshold, a dose or a citation, something is wrong and you should go and look. Agreement is much weaker evidence than it feels, because these models were trained on overlapping piles of the same internet and repeat its common errors in unison. Two models agreeing means they did not disagree, and that is all.

3. A

Notices carry two dates, and the one for downloading your data is frequently the earlier. Missing it is the expensive mistake, because the rest of the migration can be done at any point in the ninety days. Test the successor next, with your own prompts, and treat the recommended replacement as a suggestion — it was chosen for the provider's convenience and is better on their benchmarks.

4. B

Output measures — messages, words, lines of code, logins — can all be raised without producing anything, and measuring them reliably causes people to raise them. What is worth counting is work completed to a standard: time to a usable draft, minutes spent checking, tasks done at all because the barrier dropped, and errors caught before they went out. Those are harder to count, which is exactly why the easy ones get used.

5. C

Four things are worth interrupting for: a price change in your currency, a change in data terms on a tool you paste work into, a deprecation notice for something you depend on, and a capability appearing at a much smaller size or price. Everything else waits, where you will meet it with twelve prompts and a scoring sheet rather than an opinion — and reports that a tool has got lazier are exactly what the file exists to settle.

6. D

'I tried it and it was worse for this task' is a complete answer, provided you did — and it is why the twelve prompts matter, because they turn a preference into an observation. There are genuine cases for declining: where the point is that a person did it, where the struggle is the learning, where nobody can check the answer, and where somebody is owed an accountable human judgement. Declining because it is unfamiliar is not one of them.

Choosing and Using the Tools — final examination

1. A ***1. What a tool is made of***

A base model continues text rather than answering it, so a question prompts more text of the same kind. The instruct or chat suffix marks the version trained to follow instructions and hold a conversation, and that is the one almost everybody wants. People download the base model by mistake, conclude the model is broken, and give up.

2. B ***1. What a tool is made of***

Those are routing labels, not model names. It is a real answer to the question of what you are talking to: the product decides per message, the decision is invisible, and it is retuned regularly. That is the most common reason a chat app feels worse this week with no announcement — and it means your conclusions about quality will not transfer when the routing changes.

3. C ***1. What a tool is made of***

Classifying into categories you define, extracting fields from a document you supply, rewriting, translating between well-resourced languages, formatting, and answering questions about text you pasted. Where a small model falls over is knowledge it has to supply itself, long instructions with several conditions, and multi-step reasoning — because it stores less of the world and does not know that it does not know.

4. D ***1. What a tool is made of***

Visible reasoning is genuinely useful for spotting where an answer went wrong. It is not a transcript of the computation: it is text the model produced, sometimes a paraphrase generated separately, and a model can produce plausible reasoning for an answer it arrived at another way. Read it as a sketch, never as evidence of how the conclusion was reached.

5. A ***1. What a tool is made of***

Sometimes a product returns them. More often you get a convincing reconstruction — what a system prompt for an assistant like this one would probably say — which can be substantially invented. Several companies now publish their system prompts deliberately, and where they do, reading the published one explains more about the product's behaviour than any review.

6. B ***1. What a tool is made of***

Code execution is excellent for arithmetic, data files, charts and format conversion, because the answer is computed rather than recalled. Two limits catch people: no network, so nothing can be fetched or installed, and a workspace wiped between sessions and sometimes between messages. Knowing that turns a mysterious failure into a predictable one.

7. C ***1. What a tool is made of***

Your image is resized to fit a token budget and cut into tiles, so small text is the first casualty and the model reads what it can and fills in the rest. It will not flag the uncertainty. For anything where the exact characters matter, extract the text first — pdftotext if the file has a text layer, Tesseract, PaddleOCR or docTR if it does not — and let the model interpret text rather than pixels.

8. D ***2. Testing a tool yourself***

Not when the Second World War ended, but a date in a smaller conflict often confused with a bigger one; not a well-known statute, but the amendment beside it. Watching an answer drift towards the famous neighbour once teaches more about how these systems work than a week of reading. The other kinds that work are the local specific, the verifiable citation, and arithmetic with a twist.

9. A ***2. Testing a tool yourself***

The same content can take two to four times as many tokens. Three consequences arrive at once: you pay several times more for identical content, you fit proportionally less into the window, and generation is slower. It is a real and rarely acknowledged inequity in how these systems are priced, and it means a tool that handles a fifty-page English contract comfortably may truncate the same contract in Bangla.

10. B ***2. Testing a tool yourself***

Longer answers are rated higher by human and machine judges alike, largely independent of whether they are better. It is worth explicit resistance. The same applies to formatting: headings and bullet points read as thorough while being a house style set by a system prompt, and three plain sentences containing everything you needed have beaten a formatted page containing the same three sentences.

11. C ***2. Testing a tool yourself***

Arenas measure preference, not correctness, and human voters in a quick side-by-side reward answers that are longer, better formatted, more confident and more agreeable. Voters also cannot check facts in thirty seconds, so a model that is confidently wrong in a well-organised way does well — which is precisely the failure mode you most need to avoid.

12. D ***2. Testing a tool yourself***

If you cannot tell whether an answer is correct, a score of 2 means 'this looked good to me', which is exactly the judgement these tools are best at defeating. A false 2 on the prompt you understand least will quietly dominate a close comparison. Either get one person who can judge it to look, or replace it with a prompt where you can verify the answer.

13. A ***2. Testing a tool yourself***

A test set that is edited whenever something fails it stops being a test. If the prompt is genuinely badly written, rewrite it and reset the recorded history, noting that you did. The related discipline is to keep the failures — a tool getting something wrong in an interesting way becomes prompt thirteen — and to add one hard new item every six months, retiring nothing, because your prompts get clearer as you improve and clearer prompts are easier.

14. B ***2. Testing a tool yourself***

Near-deterministic is the accurate description. Tiny numerical differences can change a token, and one changed token changes everything after it. Temperature near zero is still the right setting for extraction, classification, formatting and code, because it removes most of the surprises without costing quality — but anything built on the assumption of byte-identical replies will break eventually.

15. C ***3. The routes in***

A model name, a list of role-tagged messages and a couple of settings — usually a change of base URL and a key. That compatibility is the single most valuable thing about the API route, because it makes your work portable in a field where products are withdrawn regularly. It is not complete: thinking budgets, caching controls and some tool formats need the vendor's own library.

16. D ***3. The routes in***

A key inside anything a user can reach is a public key. Requests from a browser or an app have to go through a small server of yours that holds the credential. The same logic governs sharing: a colleague creates their own account, or you use a provider or router that issues separate keys with separate limits, because a shared key means shared spending and no way to cut one person off.

17. A ***3. The routes in***

The paid API generally does not train on your inputs by default; free tiers frequently do. Experimenting with your own text is a fair trade. Experimenting with a client's contract is not, and that is a decision about somebody else's material rather than your own. Expiry and rate limits are real and are not what makes this the wrong choice.

18. B ***3. The routes in***

Most of these websites install as an app without an app store: an icon, a full-screen window, and the web version's feature set, in a few hundred kilobytes instead of three or four hundred megabytes. It also sidesteps app-store pricing, which is usually higher for the same plan, and app-store availability, which varies by country. The web version is frequently ahead on features anyway.

19. C ***3. The routes in***

Text is effectively free: a long reply is about four kilobytes, and two hundred and fifty exchanges come to roughly five megabytes. A photograph is one to four megabytes and voice is about a megabyte a minute. Resize images before uploading — a thousand-pixel-wide picture is enough for a model to read a document at a tenth of the size. A local model is one download and then nothing at all.

20. D ***3. The routes in***

Keep version control between you and the agent, and work in small commits — a change you can read in two minutes is a change you can judge, while forty rewritten files is unreviewable and people accept those because reviewing them is unpleasant. Review every line, not because the tools are bad, but because a subtly wrong function that compiles is more expensive than one that does not.

21. A ***3. The routes in***

Depending on how the page announces updates, the reader may repeat the growing answer, interrupt itself, or say nothing until the end. Some products let you turn streaming off, which usually improves matters immediately. Where none is workable, the free open front-ends are ordinary web applications and are sometimes more standards-compliant, and the API route with a plain text client is accessible by construction.

22. B ***4. What it actually costs***

One token is roughly four characters or three quarters of a word, so a million tokens is about a three-thousand-page book. These averages hold well for ordinary English prose and badly for code, tables of numbers and anything not in English. Every API response also reports exactly what you used, and free tokenisers count text before you send it.

23. C ***4. What it actually costs***

At three dollars per million that is about sixty cents a message and twelve dollars for the conversation before the model has written a word. This is where long-context bills go, and the fix is almost never a cheaper model: send less by extracting the relevant sections, start fresh conversations, and let prefix caching cover the part that genuinely must be resent each time.

24. D ***4. What it actually costs***

The discount is generally around fifty per cent for work that does not need an answer now — classifying an archive, generating catalogue descriptions, summarising last month's tickets. Two things catch people: results come back unordered, so include your own identifier with each request, and the stated window is a maximum rather than a promise, so a batch job does not belong in front of a deadline.

25. A ***4. What it actually costs***

The first three cost nothing and the last costs quality. Most people go straight to the fourth step by switching to a cheaper model everywhere, discover the drop, and conclude the work cannot be done cheaply. Cutting what you send and what you ask for is usually worth more than the model change, and it is free.

26. B ***4. What it actually costs***

Team plans charge per person per month regardless of use, and reviewing the seat list is often the largest single saving available to a small organisation. Nobody does it. The related traps are annual commitments that lock a reversible choice for a year in a field that moves every six months, spend floors on enterprise agreements, and auto-renewal after a trial.

27. C ***4. What it actually costs***

A card's foreign transaction fee is typically one to three per cent, tax on digital services is commonly fifteen to twenty per cent and often not shown until checkout, and some countries add a levy on foreign-currency payments. A twenty-dollar plan can land nearer twenty-five. Regional pricing exists in this field but is uncommon — most assistant subscriptions carry one dollar price worldwide, which is a very different proportion of income in different places.

28. D ***4. What it actually costs***

Latency is the first case: for anything a person is waiting on, doubling the wait can be the wrong trade. The second is small volume — engineering a cascade to save eighty per cent of a dollar a day is not a good use of an afternoon. The third is when you will not measure: a cascade without the hundred-item comparison is a guess that quietly ships worse work.

29. A ***5. Open models and your own hardware***

Permissive licences let you use, modify, redistribute, sell and embed, with no conditions worth worrying about beyond keeping the notice. Community licences do permit commercial use and add conditions — acceptable-use policies, passing terms downstream, naming requirements for derivatives, and in one well-known case a separate agreement above a very large user threshold. Those rarely obstruct an individual and are a real question for a business.

30. B ***5. Open models and your own hardware***

The disagreement is genuine and unresolved. Some widely used model licences fail the traditional definition because of acceptable-use restrictions or user thresholds, which is why many people say open weights instead. There is a further dispute about whether anything can be open source when the training data is not published. Different jurisdictions and bodies use the terms differently, so what matters practically is the licence text.

31. C ***5. Open models and your own hardware***

GGUF is the portable choice and runs on CPU, GPU or both on any platform, which is what you want on a laptop generally. AWQ and GPTQ are GPU serving formats used by hosting providers, with no CPU fallback. On Apple Silicon, MLX is the native option and is measurably quicker for the same model, and unified memory is why a Mac with thirty-two or sixty-four gigabytes runs models that would need an expensive card on a PC.

32. D ***5. Open models and your own hardware***

What matters on a discrete card is the amount of video memory, not the card's name, and anything that does not fit spills into system memory — not a little slower, an order of magnitude slower. The fixes in order of cheapness are to reduce the context setting, quantise the cache to eight-bit, drop one quantisation level, use a smaller model, offload some layers, and close the browser.

33. A ***5. Open models and your own hardware***

Facts degrade gently; instruction-following degrades noticeably. Watch for a model that starts ignoring parts of the instruction or drifts out of the requested format halfway through a long answer. Published perplexity figures tell you a build is two per cent worse in a statistical sense — running four of your own prompts at two quantisations for fifteen minutes tells you whether it is worse at your work.

34. B ***5. Open models and your own hardware***

The moment somebody else depends on it, you owe availability, updates, security patches and somebody to blame when it is down. The other three are the cases where local is unambiguously right: material that must not leave, enormous volume on a routine task, and permanence with no deprecation notice. The other wrong cases are hard reasoning, needing the whole product, battery life, and buying hardware to avoid a subscription.

35. C ***5. Open models and your own hardware***

Merges in particular are produced quickly, in volume, and often without evaluation of any kind. Prefer the maker's own release unless a fine-tune solves a problem you actually have, and then test it against the base model on your own prompts rather than on its card. Download counts and likes are weak signals dominated by tutorials and by whichever conversion appeared first.

36. D ***6. Data, terms and the law***

API inputs are generally not used for training by default, with retention for abuse monitoring that is commonly around thirty days and zero-retention arrangements for customers who ask and qualify. Consumer free tiers are the loosest. Paying does not automatically opt you out — several products have trained on paying subscribers' conversations with a setting available to decline.

37. A ***6. Data, terms and the law***

A model provider on the list means the product is built on somebody else's model, so your data reaches two companies and you should read both sets of terms. Where each sub-processor operates is what determines whether data leaves your region. A page that is current and specific is also telling you something about how the company is run, quite apart from what the list contains.

38. B ***6. Data, terms and the law***

A certification is meaningful evidence about organisational discipline and says nothing at all about model behaviour. It does not mean the product is safe, that the model will not make things up, or that your data cannot leak. For most small organisations it matters mainly because a customer or an insurer asks for it, and that is a legitimate reason to want it.

39. C ***6. Data, terms and the law***

Deletion going forward is real; deletion going backwards is not. Anything already absorbed into a completed training run cannot be extracted from the resulting weights, and most policies carve out backups, legal obligations and abuse records. That is why the rule that matters is applied before sending — could I email this to a stranger? — rather than after.

40. D ***6. Data, terms and the law***

A postcode, a date of birth, a sex, or a handful of location points is frequently enough, and combining a de-identified set with a public one is a documented re-identification technique. For a single document with one person in it, role substitution is genuinely effective. For a dataset of thousands, treating a removed name column as anonymisation is a mistake with regulatory consequences.

41. A ***6. Data, terms and the law***

Three questions get run together. The provider's claim is the easiest and usually answered in your favour. Whether the output is copyrightable at all turns on human authorship requirements that differ by country and are under review in several. Whether it infringes is different again and is where the real exposure sits. A provider can perfectly well assign you all their rights in something nobody has any rights in.

42. B ***6. Data, terms and the law***

A rule names a person and a moment. 'We verify AI output' is what most policies say and it is not a rule, because the failures in this area are not people ignoring an instruction — they are people believing somebody else already checked. The other four lines worth having are the approved tools, what may never be sent, whether clients are told, and who to tell when it goes wrong.

43. C ***7. Beyond the chat box***

A reference removes most of the guessing, and nearly every capable tool now accepts one. The second habit is to iterate rather than re-prompt: keep the image you nearly liked and change one thing — inpaint the hand, extend the background, adjust the light — because you are refining a result rather than rolling dice again. What separates products in practice is text rendering, subject consistency and editing an existing picture.

44. D ***7. Beyond the chat box***

A clip-on microphone beats every software choice, and two people talking at once defeats everything, which is why meeting transcripts contain nonsense. Domain vocabulary hints help noticeably — twenty expected terms is enough. Whisper is free, runs locally, and is at a standard that was commercial-grade when it appeared, so accuracy is decided before the tool is chosen.

45. A ***7. Beyond the chat box***

A few seconds of audio from a public video is enough for some tools, so keeping voices offline is not achievable for most people. The call that sounds like a family member in trouble, and the instruction that sounds like a manager approving a payment, are both defeated by a rule agreed in advance. That advice sounds excessive until it has happened to somebody you know.

46. B ***7. Beyond the chat box***

Tables are where document extraction fails most often in both routes — split across a page break, merged cells, a header repeated halfway down. Asking for output in a structured form with named fields makes a missing column visible rather than silently absorbed, and checking the totals catches the rest. Running it twice mostly tells you the tool is consistent, which it can be while being consistently wrong.

47. C ***7. Beyond the chat box***

A spreadsheet full of pasted constants is the same problem as an answer without code, and it is harder to spot because the sheet looks normal. Prefer the formula every time, even when the value is what you wanted, and apply the same preference between data tools: choose one that shows you the code, because a wrong column or a bad join is obvious in three lines of code and invisible in the chart.

48. D ***7. Beyond the chat box***

An assistant that can search everything a user can search will surface things they could always technically reach and never actually found. Organisations that deploy first discover their sharing practices through an assistant cheerfully quoting a salary spreadsheet shared with everyone in 2019. The related discipline is to pilot five seats across different roles and read the distribution rather than the average.

49. A ***7. Beyond the chat box***

Incoming mail, web pages and tickets can contain instructions aimed at the agent, and the agent carries your permissions. There is no complete fix today. The practical mitigations are least privilege, confirmation on anything consequential, and not letting one agent both read untrusted input and act on sensitive systems. Long unattended runs are a reliability problem, which short loops with checkpoints address.

50. B ***8. Living with it***

Level one is a saved prompt you paste: works everywhere, needs nothing, portable, and most people should stop there. Level two is a project or custom instruction, which is convenient and lives inside one product. Level three is a script, worth it for anything done many times a day or over more than a handful of items, because it removes the copying and pasting entirely.

51. C ***8. Living with it***

Two separate mechanisms point the same way. Models attend more reliably to the end of a long input, so the actual request is noticed. And on a paid API, prefix caching matches a byte-identical beginning, so the stable half is billed at a fraction. The ordering is right whether you are paying by the token or not.

52. D ***8. Living with it***

Most of what matters is a handful of durable facts — what you do, who you write for, your standards, your language — and those belong in a file of your own. What remains is the incidental residue of old conversations, and losing it costs less than it feels like it will. Knowing that before it becomes the reason you stay somewhere is the point.

53. A ***8. Living with it***

The saved prompt with the context and an example in it is transferable in a way that skill is not, and it is the artefact people actually use. Start from a colleague's tedious task rather than from the tool, solve one real problem visibly, and let it spread by request. A hands-on hour on the team's own work is worth running; slides are not, and neither is enthusiasm.

54. B ***8. Living with it***

An organisation that knows which of its tasks are which spends its money well; one with a headline number knows nothing, whichever direction it points. The learning curve is a real confound and so are selection effects and displaced work — time saved on drafting can reappear as time reviewing somebody else's draft, which nets to nothing organisationally while every individual honestly reports a saving.

55. C ***8. Living with it***

The honest baseline is what would actually have happened: the email sent two days later, the report that would have been shorter, the analysis nobody would have done at all. Some of the largest genuine gains sit in that last category — work that was not worth the effort before and now is — and they do not appear as minutes saved because no minutes were being spent.

56. D ***8. Living with it***

A model sounds identical when it is right and when it is inventing, and no release has changed that. Alongside it: the product layer decides your daily experience more than the model does, output costs several times input, the context window is a ceiling rather than a promise, verification cost decides whether a tool pays, and your own dozen prompts beat any published comparison. Prices, names, rankings and hardware limits all expire within months.